

First Edition, September, 1975  
Revised, October, 1975

The information in this document is  
not to be construed as a  
recommendation of Digital Equipment Corporation.  
The user assumes all responsibility  
for the use of the information in this document.  
The user assumes all responsibility  
for the use of the information in this document.  
The user assumes all responsibility  
for the use of the information in this document.  
The user assumes all responsibility  
for the use of the information in this document.

DEC-11-ORUGA-B-D

Copyright © 1975 by Digital Equipment Corporation

THE USER OF THIS SOFTWARE AGREES TO HOLD  
DIGITAL EQUIPMENT CORPORATION HARMLESS  
FROM ALL CLAIMS, DAMAGES, LOSSES AND  
EXPENSES, INCLUDING REASONABLE ATTORNEY'S  
FEES, THAT MAY BE ASSERTED AGAINST OR  
INCURRED BY DIGITAL EQUIPMENT CORPORATION  
OR ITS EMPLOYEES, AGENTS OR SUBSIDIARIES  
AS A RESULT OF THE USE OF THIS SOFTWARE.

The following are trademarks of Digital Equipment Corporation:

|           |           |           |           |           |           |           |           |           |           |           |
|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|
| DEC-11    | DEC-10    | DEC-9     | DEC-8     | DEC-7     | DEC-6     | DEC-5     | DEC-4     | DEC-3     | DEC-2     | DEC-1     |
| 11-000000 | 10-000000 | 09-000000 | 08-000000 | 07-000000 | 06-000000 | 05-000000 | 04-000000 | 03-000000 | 02-000000 | 01-000000 |
| 11-000000 | 10-000000 | 09-000000 | 08-000000 | 07-000000 | 06-000000 | 05-000000 | 04-000000 | 03-000000 | 02-000000 | 01-000000 |
| 11-000000 | 10-000000 | 09-000000 | 08-000000 | 07-000000 | 06-000000 | 05-000000 | 04-000000 | 03-000000 | 02-000000 | 01-000000 |
| 11-000000 | 10-000000 | 09-000000 | 08-000000 | 07-000000 | 06-000000 | 05-000000 | 04-000000 | 03-000000 | 02-000000 | 01-000000 |
| 11-000000 | 10-000000 | 09-000000 | 08-000000 | 07-000000 | 06-000000 | 05-000000 | 04-000000 | 03-000000 | 02-000000 | 01-000000 |
| 11-000000 | 10-000000 | 09-000000 | 08-000000 | 07-000000 | 06-000000 | 05-000000 | 04-000000 | 03-000000 | 02-000000 | 01-000000 |
| 11-000000 | 10-000000 | 09-000000 | 08-000000 | 07-000000 | 06-000000 | 05-000000 | 04-000000 | 03-000000 | 02-000000 | 01-000000 |
| 11-000000 | 10-000000 | 09-000000 | 08-000000 | 07-000000 | 06-000000 | 05-000000 | 04-000000 | 03-000000 | 02-000000 | 01-000000 |
| 11-000000 | 10-000000 | 09-000000 | 08-000000 | 07-000000 | 06-000000 | 05-000000 | 04-000000 | 03-000000 | 02-000000 | 01-000000 |
| 11-000000 | 10-000000 | 09-000000 | 08-000000 | 07-000000 | 06-000000 | 05-000000 | 04-000000 | 03-000000 | 02-000000 | 01-000000 |
| 11-000000 | 10-000000 | 09-000000 | 08-000000 | 07-000000 | 06-000000 | 05-000000 | 04-000000 | 03-000000 | 02-000000 | 01-000000 |

Order additional copies as directed on the Software  
Information page at the back of this document.

digital equipment corporation maynard, massachusetts



First Printing, September, 1973  
Revised, October, 1974

The information in this document is subject to change without notice and should not be construed as a contract of Digital Equipment Corporation. Digital Equipment Corporation assumes no responsibility for any errors that may appear here.

The software described herein can be copied under a license for use in such systems, excluding any system, except as may be permitted by DIGITAL.

Digital Equipment Corporation assumes no responsibility for the use or reliability of its software on equipment that is not supplied by DIGITAL.

Copyright © 1973, 1974 by Digital Equipment Corporation

The HOW TO OBTAIN SOFTWARE INFORMATION page, located at the back of this document, explains the various services available to DIGITAL software users.

The postage prepaid READER'S COMMENTS form on the last page of this document requests the user's critical evaluation to assist us in preparing future documentation.

The following are trademarks of Digital Equipment Corporation:

|              |           |         |            |
|--------------|-----------|---------|------------|
| CDP          | DIGITAL   | INDAC   | PS/8       |
| COMPUTER LAB | DNC       | KA10    | QUICKPOINT |
| COMSYST      | EDGRIN    | LAB-8   | SD-8       |
| COMTEX       | EDUSYSTEM | LAB-8/e | SS         |
| DDT          | FLIP CHIP | LAB-K   | RSX        |
| DEC          | FOCAL     | OMNIBUS | RTM        |
| DECCOMM      | GLC-8     | OS/8    | RT-11      |
| DECTAPE      | IDAC      | PDP     | SABR       |
| DIBOL        | IDACS     | PHA     | TYPESET 8  |
|              |           |         | UNIBUS     |



## CONTENTS

|  | <u>Page</u> |
|--|-------------|
| PREFACE                                      | xiv         |
| CHAPTER 1                                    |             |
| RT-11 OVERVIEW                               | 1-1         |
| 1.1 PROGRAM DEVELOPMENT                      | 1-2         |
| 1.2 SYSTEM SOFTWARE COMPONENTS               | 1-3         |
| 1.3 SYSTEM HARDWARE COMPONENTS               | 1-5         |
| 1.4 USING THE RT-11 SYSTEM                   | 1-6         |
| 1.4.1 RT-11 Single-Job Monitor               | 1-6         |
| 1.4.2 RT-11 Foreground/Background Monitor    | 1-6         |
| 1.4.3 Facilities Available Only in RT-11 F/B | 1-7         |
| CHAPTER 2                                    |             |
| SYSTEM COMMUNICATION                         | 2-1         |
| 2.1 START PROCEDURE                          | 2-1         |
| 2.2 SYSTEM CONVENTIONS                       | 2-3         |
| 2.2.1 Data Formats                           | 2-3         |
| 2.2.2 Prompting Characters                   | 2-4         |
| 2.2.3 Physical Device Names                  | 2-4         |
| 2.2.4 File Names and Extensions              | 2-5         |
| 2.3 MONITOR SOFTWARE COMPONENTS              | 2-7         |
| 2.3.1 Resident Monitor (RMON)                | 2-7         |
| 2.3.2 Keyboard Monitor (KMON)                | 2-7         |
| 2.3.3 User Service Routine (USR)             | 2-7         |
| 2.3.4 Device Handlers                        | 2-8         |
| 2.4 GENERAL MEMORY LAYOUT                    | 2-8         |
| 2.4.1 Component Sizes                        | 2-9         |
| 2.5 ENTERING I/O INFORMATION                 | 2-10        |
| 2.6 KEYBOARD COMMUNICATION (KMON)            | 2-11        |
| 2.6.1 Foreground/Background Terminal I/O     | 2-13        |
| 2.6.2 Type-Ahead                             | 2-14        |
| 2.7 KEYBOARD COMMANDS                        | 2-14        |
| 2.7.1 Commands to Control Terminal I/O       | 2-15        |
| (GT ON and GT OFF)                           |             |



|           |  |      |
|-----------|--|------|
| 2.7.2     | Commands to Allocate System Resources                        | 2-16 |
| 2.7.2.1   | DATE Command   | 2-16 |
| 2.7.2.2   | TIME Command   | 2-17 |
| 2.7.2.3   | INITIALIZE Command   | 2-18 |
| 2.7.2.4   | ASSIGN Command   | 2-18 |
| 2.7.2.5   | CLOSE Command  | 2-20 |
| 2.7.2.6   | LOAD Command   | 2-20 |
| 2.7.2.7   | UNLOAD Command   | 2-21 |
| 2.7.2.8   | SET Command  | 2-23 |
| 2.7.3     | Commands to Manipulate Memory Images                         | 2-27 |
| 2.7.3.1   | GET Command  | 2-27 |
| 2.7.3.2   | Base Command   | 2-28 |
| 2.7.3.3   | Examine Command  | 2-29 |
| 2.7.3.4   | Deposit Command  | 2-29 |
| 2.7.3.5   | SAVE Command   | 2-30 |
| 2.7.4     | Commands to Start a Program                                  | 2-32 |
| 2.7.4.1   | RUN Command  | 2-32 |
| 2.7.4.2   | R Command  | 2-33 |
| 2.7.4.3   | START Command  | 2-33 |
| 2.7.4.4   | REENTER Command  | 2-34 |
| 2.7.5     | Commands Used Only in a<br>Foreground/Background Environment | 2-34 |
| 2.7.5.1   | FRUN Command   | 2-35 |
| 2.7.5.2   | SUSPEND Command  | 2-36 |
| 2.7.5.3   | RSUME Command  | 2-37 |
| 2.8       | MONITOR ERROR MESSAGES                                       | 2-37 |
| 2.8.1     | Monitor HALTS  | 2-40 |
| CHAPTER 3 | TEXT EDITOR  | 3-1  |
| 3.1       | CALLING AND USING EDIT                                       | 3-1  |
| 3.2       | MODES OF OPERATION   | 3-2  |
| 3.3       | SPECIAL KEY COMMANDS   | 3-2  |
| 3.4       | COMMAND STRUCTURE  | 3-3  |
| 3.4.1     | Arguments  | 3-4  |
| 3.4.2     | Command Strings  | 3-5  |
| 3.4.3     | The Current Location Pointer                                 | 3-6  |
| 3.4.4     | Character- and Line-Oriented<br>Command Properties           | 3-6  |
| 3.4.5     | Command Repetition   | 3-8  |
| 3.5       | MEMORY USAGE   | 3-9  |
| 3.6       | EDITING COMMANDS   | 3-10 |
| 3.6.1     | Input/Output Commands  | 3-10 |
| 3.6.1.1   | Edit Read  | 3-10 |
| 3.6.1.2   | Edit Write   | 3-11 |
| 3.6.1.3   | Edit Backup  | 3-11 |
| 3.6.1.4   | Read   | 3-12 |
| 3.6.1.5   | Write  | 3-13 |
| 3.6.1.6   | Next   | 3-14 |



|           |  |      |
|-----------|--|------|
| 3.6.1.7   | List                                     | 3-14 |
| 3.6.1.8   | Verify                                   | 3-15 |
| 3.6.1.9   | End File                                 | 3-15 |
| 3.6.2     | Pointer Relocation Commands              | 3-16 |
| 3.6.2.1   | Beginning                                | 3-16 |
| 3.6.2.2   | Jump                                     | 3-17 |
| 3.6.2.3   | Advance                                  | 3-17 |
| 3.6.3     | Search Commands                          | 3-18 |
| 3.6.3.1   | Get                                      | 3-18 |
| 3.6.3.2   | Find                                     | 3-19 |
| 3.6.3.3   | Position                                 | 3-20 |
| 3.6.4     | Text Modification Commands               | 3-20 |
| 3.6.4.1   | Insert                                   | 3-20 |
| 3.6.4.2   | Delete                                   | 3-   |
| 3.6.4.3   | Kill                                     |      |
| 3.6.4.4   | Change                                   |      |
| 3.6.4.5   | Exchange                                 |      |
| 3.6.5     | Utility Commands                         |      |
| 3.6.5.1   | Save                                     | 3-24 |
| 3.6.5.2   | Unsave                                   | 3-25 |
| 3.6.5.3   | Macro                                    | 3-25 |
| 3.6.5.4   | Execute Macro                            | 3-26 |
| 3.6.5.5   | Edit Version                             | 3-27 |
| 3.7       | THE DISPLAY EDITOR                       | 3-27 |
| 3.7.1     | Using the Display Editor                 | 3-28 |
| 3.7.2     | Setting the Editor to Immediate Mode     | 3-30 |
| 3.8       | EDIT EXAMPLE                             | 3-32 |
| 3.9       | EDIT ERROR MESSAGES                      | 3-33 |
| CHAPTER 4 | PERIPHERAL INTERCHANGE PROGRAM (PIP)     | 4-1  |
| 4.1       | CALLING AND USING PIP                    | 4-1  |
| 4.1.1     | Using the "Wild Card" Construction       | 4-1  |
| 4.2       | PIP SWITCHES                             | 4-2  |
| 4.2.1     | Operations Involving Magtape or Cassette | 4-4  |
| 4.2.2     | Copy Operations                          | 4-9  |
| 4.2.3     | Multiple Copy Operations                 | 4-11 |
| 4.2.4     | The Extend and Delete Operations         | 4-13 |
| 4.2.5     | The Rename Operation                     | 4-15 |
| 4.2.6     | Directory List Operations                | 4-15 |
| 4.2.7     | The Directory Initialization Operation   | 4-18 |
| 4.2.8     | The Compress Operation                   | 4-19 |
| 4.2.9     | The Bootstrap Copy Operation             | 4-20 |
| 4.2.10    | The Boot Operation                       | 4-20 |
| 4.2.11    | The Version Switch                       | 4-21 |
| 4.2.12    | Bad Block Scan (/K)                      | 4-21 |
| 4.2.12.1  | Recovery from Bad Blocks                 | 4-21 |
| 4.3       | PIP ERROR MESSAGES                       | 4-24 |



|           |   |      |
|-----------|---|------|
| CHAPTER 5 | MACRO ASSEMBLER                         | 5-1  |
| 5.1       | SOURCE PROGRAM FORMAT                   | 5-2  |
| 5.1.1     | Statement Format                        | 5-2  |
| 5.1.1.1   | Label Field                             | 5-3  |
| 5.1.1.2   | Operator Field                          | 5-3  |
| 5.1.1.3   | Operand Field                           | 5-4  |
| 5.1.1.4   | Comment Field                           | 5-4  |
| 5.1.2     | Format Control                          | 5-5  |
| 5.2       | SYMBOLS AND EXPRESSIONS                 | 5-5  |
| 5.2.1     | Character Set                           | 5-5  |
| 5.2.1.1   | Separating and Delimiting Characters    | 5-6  |
| 5.2.1.2   | Illegal Characters                      | 5-7  |
| 5.2.1.3   | Operator Characters                     | 5-8  |
| 5.2.2     | Symbols                                 | 5-9  |
| 5.2.2.1   | Permanent Symbols                       | 5-9  |
| 5.2.2.2   | User-Defined and Macro Symbols          | 5-9  |
| 5.2.3     | Direct Assignment                       | 5-10 |
| 5.2.4     | Register Symbols                        | 5-11 |
| 5.2.5     | Local Symbols                           | 5-12 |
| 5.2.6     | Assembly Location Counter               | 5-14 |
| 5.2.7     | Numbers                                 | 5-17 |
| 5.2.8     | Terms                                   | 5-17 |
| 5.2.9     | Expressions                             | 5-18 |
| 5.3       | RELOCATION AND LINKING                  | 5-19 |
| 5.4       | ADDRESSING MODES                        | 5-20 |
| 5.4.1     | Register Mode                           | 5-21 |
| 5.4.2     | Register Deferred Mode                  | 5-21 |
| 5.4.3     | Autoincrement Mode                      | 5-21 |
| 5.4.4     | Autoincrement Deferred Mode             | 5-22 |
| 5.4.5     | Autodecrement Mode                      | 5-23 |
| 5.4.6     | Autodecrement Deferred Mode             | 5-23 |
| 5.4.7     | Index Mode                              | 5-23 |
| 5.4.8     | Index Deferred Mode                     | 5-23 |
| 5.4.9     | Immediate Mode                          | 5-24 |
| 5.4.10    | Absolute Mode                           | 5-24 |
| 5.4.11    | Relative Mode                           | 5-24 |
| 5.4.12    | Relative Deferred Mode                  | 5-25 |
| 5.4.13    | Table of Mode Forms and Codes           | 5-25 |
| 5.4.14    | Branch Instruction Addressing           | 5-26 |
| 5.4.15    | EMT and TRAP Addressing                 | 5-27 |
| 5.5       | ASSEMBLER DIRECTIVES                    | 5-27 |
| 5.5.1     | Listing Control Directives              | 5-27 |
| 5.5.1.1   | .LIST and .NLIST                        | 5-27 |
| 5.5.1.2   | Page Headings                           | 5-34 |
| 5.5.1.3   | .TITLE                                  | 5-34 |
| 5.5.1.4   | .SBTTL                                  | 5-34 |
| 5.5.1.5   | .IDENT                                  | 5-36 |
| 5.5.1.6   | Page Ejection                           | 5-36 |
| 5.5.2     | Functions: .ENABL and .DSABL Directives | 5-36 |
| 5.5.3     | Data Storage Directives                 | 5-37 |



|          |   |      |
|----------|---|------|
| 5.5.3.1  | .BYTE   | 5-38 |
| 5.5.3.2  | .WORD   | 5-39 |
| 5.5.3.3  | ASCII Conversion of One or Two Characters                               | 5-40 |
| 5.5.3.4  | .ASCII  | 5-41 |
| 5.5.3.5  | .ASCIZ  | 5-42 |
| 5.5.3.6  | .RAD50  | 5-43 |
| 5.5.4    | Radix Control   | 5-44 |
| 5.5.4.1  | .RADIX  | 5-44 |
| 5.5.4.2  | Temporary Radix Control: $\uparrow D$ , $\uparrow O$ , and $\uparrow B$ | 5-45 |
| 5.5.5    | Location Counter Control  | 5-46 |
| 5.5.5.1  | .EVEN   | 5-46 |
| 5.5.5.2  | .ODD  | 5-46 |
| 5.5.5.3  | .BLKB and .BLKW   | 5-47 |
| 5.5.6    | Numeric Control   | 5-47 |
| 5.5.6.1  | .FLT2 and .FLT4   | 5-48 |
| 5.5.6.2  | Temporary Numeric Control: $\uparrow F$ and $\uparrow C$                | 5-49 |
| 5.5.7    | Terminating Directives  | 5-50 |
| 5.5.7.1  | .END  | 5-50 |
| 5.5.7.2  | .EOT  | 5-51 |
| 5.5.8    | Program Boundaries Directive: .LIMIT                                    | 5-51 |
| 5.5.9    | Program Section Directives  | 5-51 |
| 5.5.10   | Symbol Control: .GLOBL  | 5-54 |
| 5.5.11   | Conditional Assembly Directives   | 5-55 |
| 5.5.11.1 | Subconditionals   | 5-57 |
| 5.5.11.2 | Immediate Conditionals  | 5-58 |
| 5.5.11.3 | PAL-11R and PAL-11S Conditional Assembly Directives                     | 5-59 |
| 5.6      | MACRO DIRECTIVES  |      |
| 5.6.1    | Macro Definition  | 5-60 |
| 5.6.1.1  | .MACRO  | 5-60 |
| 5.6.1.2  | .ENDM   | 5-60 |
| 5.6.1.3  | .MEXIT  | 5-61 |
| 5.6.1.4  | MACRO Definition Formatting   | 5-61 |
| 5.6.2    | Macro Calls   | 5-62 |
| 5.6.3    | Arguments to Macro Calls and Definitions                                | 5-62 |
| 5.6.3.1  | Macro Nesting   | 5-63 |
| 5.6.3.2  | Special Characters  | 5-64 |
| 5.6.3.3  | Numeric Arguments Passed as Symbols                                     | 5-64 |
| 5.6.3.4  | Number of Arguments   | 5-65 |
| 5.6.3.5  | Automatically Created Symbols Within User-Defined Macros                | 5-66 |
| 5.6.3.6  | Concatenation   | 5-67 |
| 5.6.4    | .NARG, .NCHR, and .NTYPE  | 5-68 |
| 5.6.5    | .ERROR and .PRINT   | 5-70 |
| 5.6.6    | Indefinite Repeat Block: .IRP and .IRPC                                 | 5-71 |
| 5.6.7    | Repeat Block: .REPT   | 5-73 |
| 5.6.8    | Macro Libraries: .MCALL   | 5-74 |
| 5.7      | CALLING AND USING MACRO   |      |
| 5.7.1    | Switches  | 5-74 |
| 5.7.1.1  | Listing Control Switches  | 5-76 |
| 5.7.1.2  | Function Switches   | 5-77 |
| 5.7.1.3  | Cross Reference Table Generation (CREF)                                 | 5-78 |
| 5.8      | MACRO ERROR MESSAGES  | 5-84 |



|           |   |      |
|-----------|---|------|
| CHAPTER 6 | LINKER                                    | 6-1  |
| 6.1       | INTRODUCTION                              | 6-1  |
| 6.2       | CALLING AND USING THE LINKER              | 6-2  |
| 6.2.1     | Command String                            | 6-2  |
| 6.2.2     | Switches                                  | 6-3  |
| 6.3       | ABSOLUTE AND RELOCATABLE PROGRAM SECTIONS | 6-4  |
| 6.4       | GLOBAL SYMBOLS                            | 6-5  |
| 6.5       | INPUT AND OUTPUT                          | 6-5  |
| 6.5.1     | Object Modules                            | 6-5  |
| 6.5.2     | Load Module                               | 6-5  |
| 6.5.3     | Load Map                                  | 6-7  |
| 6.5.4     | Library Files                             | 6-8  |
| 6.6       | USING OVERLAYS                            | 6-10 |
| 6.7       | USING LIBRARIES                           | 6-15 |
| 6.7.1     | User Library Searches                     | 6-16 |
| 6.8       | SWITCH DESCRIPTION                        | 6-18 |
| 6.8.1     | Alphabetize Switch                        | 6-18 |
| 6.8.2     | Bottom Address Switch                     | 6-18 |
| 6.8.3     | Continue Switch                           | 6-20 |
| 6.8.4     | Default FORTRAN Library Switch            | 6-20 |
| 6.8.5     | Include Switch                            | 6-20 |
| 6.8.6     | LDA Format Switch                         | 6-21 |
| 6.8.7     | Modify Stack Address                      | 6-21 |
| 6.8.8     | Overlay Switch                            | 6-21 |
| 6.8.9     | REL Format Switch                         | 6-23 |
| 6.8.10    | Symbol Table Switch                       | 6-23 |
| 6.8.11    | Transfer Address Switch                   | 6-24 |
| 6.9       | LINKER ERROR HANDLING AND MESSAGES        | 6-24 |
| CHAPTER 7 | LIBRARIAN                                 | 7-1  |
| 7.1       | CALLING AND USING LIBR                    | 7-1  |
| 7.2       | USER SWITCH COMMANDS AND FUNCTIONS        | 7-2  |
| 7.2.1     | Command Syntax                            | 7-2  |
| 7.2.2     | LIBR Switch Commands                      | 7-2  |
| 7.2.2.1   | Command Continuation Switch               | 7-3  |
| 7.2.2.2   | Creating a Library File                   | 7-4  |
| 7.2.2.3   | Inserting Modules Into a Library          | 7-5  |
| 7.2.2.4   | Replace Switch                            | 7-5  |
| 7.2.2.5   | Delete Switch                             | 7-6  |
| 7.2.2.6   | Delete Global Switch                      | 7-7  |
| 7.2.2.7   | Update Switch                             | 7-9  |
| 7.2.2.8   | Listing the Directory of a Library File   | 7-9  |



|           |   |      |
|-----------|---|------|
| 7.2.2.9   | Merging Library Files                       | 7-10 |
| 7.3       | COMBINING LIBRARY SWITCH FUNCTIONS          | 7-11 |
| 7.4       | FORMAT OF LIBRARY FILES                     | 7-12 |
| 7.4.1     | Library Header                              | 7-12 |
| 7.4.2     | Entry Point Table (Library Directory)       | 7-13 |
| 7.4.3     | Object Modules                              | 7-14 |
| 7.4.4     | Library End Trailer                         | 7-14 |
| 7.5       | LIBR ERROR MESSAGES                         | 7-14 |
| CHAPTER 8 | ON-LINE DEBUGGING TECHNIQUE                 | 8-1  |
| 8.1       | CALLING AND USING ODT                       | 8-1  |
| 8.1.1     | Return to Monitor, CTRL C                   | 8-3  |
| 8.1.2     | Terminate Search, CTRL U                    | 8-4  |
| 8.2       | RELOCATION                                  | 8-4  |
| 8.2.1     | Relocatable Expressions                     | 8-4  |
| 8.3       | COMMANDS AND FUNCTIONS                      | 8-5  |
| 8.3.1     | Printout Formats                            | 8-5  |
| 8.3.2     | Opening, Changing and Closing Locations     | 8-6  |
| 8.3.3     | Accessing General Registers 0-7             | 8-9  |
| 8.3.4     | Accessing Internal Registers                | 8-10 |
| 8.3.5     | Radix 50 Mode, X                            | 8-10 |
| 8.3.6     | Breakpoints                                 | 8-11 |
| 8.3.7     | Running the Program, r;G and r;P            | 8-12 |
| 8.3.8     | Single-Instruction Mode                     | 8-14 |
| 8.3.9     | Searches                                    | 8-14 |
| 8.3.10    | The Constant Register, r;C                  | 8-16 |
| 8.3.11    | Memory Block Initialization, ;F and ;I      | 8-16 |
| 8.3.12    | Calculating Offsets, r;O                    | 8-17 |
| 8.3.13    | Relocation Register Commands, r;nR, ;nR, ;R | 8-17 |
| 8.3.14    | The Relocation Calculators, nR and nI       | 8-18 |
| 8.3.15    | ODT Priority Level, \$P                     | 8-19 |
| 8.3.16    | ASCII Input and Output, r;nA                | 8-20 |
| 8.4       | PROGRAMMING CONSIDERATIONS                  | 8-20 |
| 8.4.1     | Functional Organization                     | 8-20 |
| 8.4.2     | Breakpoints                                 | 8-21 |
| 8.4.3     | Searches                                    | 8-24 |
| 8.4.4     | Terminal Interrupt                          | 8-24 |
| 8.5       | ODT ERROR DETECTION                         | 8-25 |
| CHAPTER 9 | PROGRAMMED REQUESTS                         | 9-1  |
| 9.1       | FORMAT OF A PROGRAMMED REQUEST              | 9-2  |
| 9.2       | SYSTEM CONCEPTS                             | 9-5  |
| 9.2.1     | Channel Number (chan)                       | 9-5  |
| 9.2.2     | Device Block (dbl)                          | 9-5  |
| 9.2.3     | EMT Argument Blocks                         | 9-5  |



|         |                                |      |
|---------|--------------------------------|------|
| 9.2.4   | Important Memory Areas         | 9-6  |
| 9.2.4.1 | Vector Addresses               | 9-6  |
| 9.2.4.2 | Resident Monitor               | 9-7  |
| 9.2.4.3 | System Communication Area      | 9-7  |
| 9.2.5   | Swapping Algorithm             | 9-9  |
| 9.2.6   | Offset Words                   | 9-11 |
| 9.2.7   | File Structure                 | 9-13 |
| 9.2.8   | Completion Routines            | 9-13 |
| 9.2.9   | Using the System Macro Library | 9-14 |
| 9.3     | TYPES OF PROGRAMMED REQUESTS   | 9-14 |
| 9.3.1   | System Macros                  | 9-20 |
| 9.3.1.1 | .DATE                          | 9-20 |
| 9.3.1.2 | .INTEN                         | 9-21 |
| 9.3.1.3 | .REGDEF                        | 9-22 |
| 9.3.1.4 | .SYNCH                         | 9-22 |
| 9.3.1.5 | ..V2..                         | 9-24 |
| 9.4     | PROGRAMMED REQUEST USAGE       | 9-25 |
| 9.4.1   | .CDFN                          | 9-25 |
| 9.4.2   | .CHAIN                         | 9-27 |
| 9.4.3   | .CHCOPY                        | 9-28 |
| 9.4.4   | .CLOSE                         | 9-30 |
| 9.4.5   | .CMKT                          | 9-31 |
| 9.4.6   | .CNTXSW                        | 9-32 |
| 9.4.7   | .CSIGEN                        | 9-33 |
| 9.4.8   | .CSISPC                        | 9-36 |
| 9.4.8.1 | Passing Switch Information     | 9-38 |
| 9.4.9   | .CSTAT                         | 9-41 |
| 9.4.10  | .DELETE                        | 9-42 |
| 9.4.11  | .DEVICE                        | 9-44 |
| 9.4.12  | .DSTATUS                       | 9-45 |
| 9.4.13  | .ENTER                         | 9-47 |
| 9.4.14  | .EXIT                          | 9-49 |
| 9.4.15  | .FETCH                         | 9-50 |
| 9.4.16  | .GTIM                          | 9-51 |
| 9.4.17  | .GTJB                          | 9-52 |
| 9.4.18  | .HERR/.SERR                    | 9-53 |
| 9.4.19  | .HRESET                        | 9-55 |
| 9.4.20  | .LOCK/.UNLOCK                  | 9-56 |
| 9.4.21  | .LOOKUP                        | 9-58 |
| 9.4.22  | .MRKT                          | 9-60 |
| 9.4.23  | .MWAIT                         | 9-62 |
| 9.4.24  | .PRINT                         | 9-63 |
| 9.4.25  | .PROTECT                       | 9-64 |
| 9.4.26  | .PURGE                         | 9-65 |
| 9.4.27  | .QSET                          | 9-65 |
| 9.4.28  | .RCTRLO                        | 9-67 |
| 9.4.29  | .RCVD/.RCVDW/.RCVDC            | 9-68 |
| 9.4.30  | .READ/.READC/.READW            | 9-71 |
| 9.4.31  | .RELEAS                        | 9-74 |
| 9.4.32  | .RENAME                        | 9-75 |
| 9.4.33  | .REOPEN                        | 9-77 |
| 9.4.34  | .SAVESTATUS                    | 9-77 |
| 9.4.35  | .SDAT/.SDATC/.SDATW            | 9-80 |



|            |   |       |
|------------|---|-------|
| 9.4.36     | .SETTOP   | 9-82  |
| 9.4.37     | .SFPA   | 9-84  |
| 9.4.38     | .SPFUN  | 9-85  |
| 9.4.39     | .SPND/.RSUM   | 9-87  |
| 9.4.40     | .SRESET   | 9-90  |
| 9.4.41     | .TLOCK  | 9-91  |
| 9.4.42     | .TRPSET   | 9-92  |
| 9.4.43     | .TTYIN/.TTINR   | 9-93  |
| 9.4.44     | .TTYOUT/.TTOUTR   | 9-95  |
| 9.4.45     | .TWAIT  | 9-98  |
| 9.4.46     | .WAIT   | 9-99  |
| 9.4.47     | .WRITE/.WRITC/.WRITW  | 9-100 |
| 9.5        | CONVERTING VERSION 1 MACRO CALLS<br>TO VERSION 2              | 9-108 |
| 9.5.1      | Macro Calls Requiring No Conversion                           | 9-108 |
| 9.5.2      | Macro Calls Which May Be Converted                            | 9-108 |
| CHAPTER 10 | EXPAND UTILITY PROGRAM  | 10-1  |
| 10.1       | LANGUAGE  | 10-1  |
| 10.2       | RESTRICTIONS  | 10-1  |
| 10.3       | CALLING AND USING EXPAND                                      | 10-2  |
| 10.4       | EXPAND ERROR MESSAGES   | 10-6  |
| CHAPTER 11 | ASEMBL, THE 8K ASSEMBLER                                      | 11-1  |
| 11.1       | CALLING AND USING ASEMBL                                      | 11-1  |
| 11.2       | ASEMBL ERROR MESSAGES   | 11-6  |
| APPENDIX A | ASSEMBLY, LINK, AND BUILD INSTRUCTIONS                        | A-1   |
| APPENDIX B | COMMAND AND SWITCH SUMMARIES                                  | B-1   |
| APPENDIX C | MACRO ASSEMBLER, INSTRUCTION, AND<br>CHARACTER CODE SUMMARIES | C-1   |
| APPENDIX D | SYSTEM MACRO FILE   | D-1   |
| APPENDIX E | PROGRAMMED REQUEST SUMMARY                                    | E-1   |
| APPENDIX F | BASIC/RT-11 LANGUAGE SUMMARY                                  | F-1   |
| APPENDIX G | FORTRAN LANGUAGE SUMMARY                                      | G-1   |
| APPENDIX H | F/B PROGRAMMING AND DEVICE HANDLERS                           | H-1   |
| APPENDIX I | DUMP  | I-1   |
| APPENDIX J | FILEX   | J-1   |



|            |                         |            |
|------------|-------------------------|------------|
| APPENDIX K | SOURCE COMPARE (SRCCOM) | K-1        |
| APPENDIX L | PATCH                   | L-1        |
| APPENDIX M | PATCHO                  | M-1        |
| APPENDIX N | DISPLAY FILE HANDLER    | N-1        |
| APPENDIX O | BATCH                   | O-1        |
| APPENDIX P | ERROR MESSAGE SUMMARY   | P-1        |
| GLOSSARY   |                         | GLOSSARY-1 |
| INDEX      |                         | INDEX-1    |

## TABLES

| <u>Number</u> |                                    | <u>Page</u> |
|---------------|------------------------------------|-------------|
| 2-1           | Prompting Characters               | 2-4         |
| 2-2           | Physical Device Names              | 2-5         |
| 2-3           | File Name Extensions               | 2-6         |
| 2-4           | Special Function Keys              | 2-12        |
| 2-5           | SET Command Options                | 2-23        |
| 3-1           | EDIT Key Commands                  | 3-2         |
| 3-2           | Command Arguments                  | 3-5         |
| 3-3           | Immediate Mode Commands            | 3-30        |
| 4-1           | PIP Switches                       | 4-3         |
| 5-1           | Legal Separating Characters        | 5-6         |
| 6-1           | Linker Switches                    | 6-3         |
| 7-1           | LIBR Switches                      | 7-3         |
| 8-1           | Forms of Relocatable Expressions   | 8-5         |
| 8-2           | Internal Registers                 | 8-10        |
| 8-3           | Radix 50 Terminators               | 8-11        |
| 9-1           | Summary of Programmed Requests     | 9-14        |
| 9-2           | Requests Requiring the USR         | 9-19        |
| 11-1          | Directives not Available in ASEMBL | 11-2        |
| A-1           | CROOT (QCBOOT) Instructions        | A-7         |



|     |                      |     |
|-----|----------------------|-----|
| I-1 | DUMP Switches        | I-2 |
| J-1 | FILEX Switch Options | J-2 |
| K-1 | SRCCOM Switches      | K-2 |
| L-1 | PATCH Commands       | L-2 |

## FIGURES

| <u>Number</u> |  | <u>Page</u> |
|---------------|--|-------------|
| 2-1           | RT-11 System Memory Maps   | 2-8         |
| 2-2           | RT-11 Memory Map (GT40)  | 2-9         |
| 3-1           | Display Editor Format  | 3-28        |
| 5-1           | Assembly Source Listing of MACRO<br>Code Showing Local Symbol Blocks | 5-15        |
| 5-2           | Example of MACRO Line Printer<br>Listing (132-column Line Printer)   | 5-31        |
| 5-3           | Example of Page Heading From<br>MACRO (80-column Line Printer)       | 5-32        |
| 5-4           | Symbol Table   | 5-33        |
| 5-5           | Assembly Listing Table of Contents                                   | 5-35        |
| 5-6           | .IRP and .IRPC Example   | 5-73        |
| 5-7           | MACRO Source Code  | 5-80        |
| 5-8           | CREF Listing Output  | 5-81        |
| 6-1           | Linker Load Map for Background Job                                   | 6-9         |
| 6-2           | Overlay Scheme   | 6-10        |
| 6-3           | Memory Diagram Showing BASIC<br>Link with Overlay Regions            | 6-11        |
| 6-4           | Run-Time Overlay Handler   | 6-12        |
| 6-5           | Library Searches   | 6-17        |
| 6-6           | Alphabetized Load Map for a<br>Background Job                        | 6-19        |
| 7-1           | General Library File Format  | 7-12        |
| 7-2           | Library Header Format  | 7-13        |
| 7-3           | Format of Entry Point Table  | 7-13        |
| 7-4           | Library End Trailer  | 7-14        |



1. The first part of the report is a general  
description of the project and its objectives.  
2. The second part is a detailed description of the  
methodology used in the study.  
3. The third part is a description of the results  
of the study.  
4. The fourth part is a discussion of the results  
and their implications.  
5. The fifth part is a conclusion and a list of  
references.



## PREFACE

This manual describes the use of the RT-11 Operating System, including both Single-Job and Foreground/Background monitors. It assumes that the user has had some exposure to assembly language programming and computer systems in general. For definitions of the technical terminology used within the documentation, consult the Glossary at the end of the manual. In addition, the introductory chapters of the PDP-11 PAPER TAPE SOFTWARE PROGRAMMING HANDBOOK (DEC-11-XPTSA-A-D), the PDP-11 PROCESSOR HANDBOOK, and the PDP-11 PERIPHERALS HANDBOOK can also be consulted for reference purposes. The "Getting Started With RT-11" document (DEC-11-ORCPA-D-D) included with each system provides a basic introduction to RT-11; by following the instructions in its demonstration, the user can become acquainted with RT-11 operating procedures.

Upon receiving the RT-11 system it is recommended that the user first read through the entire manual to become familiar with system conventions. The concepts of the system will become clear if the manual is read in its entirety first for a general introduction to RT-11 as a whole, and then reread for specific information.

Chapters 1 and 2 discuss general system operations and monitor keyboard commands used to control jobs and to implement user programs. System programs (EDIT, PIP, MACRO, LINKER, LIBR, ODT) are described in Chapters 3 through 8. Chapter 9 describes programmed requests; this chapter will be of particular interest to the experienced programmer who wishes to make use of monitor services in his own assembly language programs. Chapters 10 and 11 explain the 8K Assembler and EXPAND programs which are most useful in RT-11 systems with minimum memory configurations. The appendixes summarize the contents of the manual and introduce several additional utility programs used for various non-standard system operations (FILEX, SRCCOM, PATCH, DUMP).

All programs discussed in the manual may be used in any Single-job or Foreground/Background environment with the exception of the MACRO assembler (Chapter 5), which requires a minimum of 12K of memory for execution, and with the exception of certain monitor commands that can be used only in a F/B environment (see Chapter 2, Section 2.7.5).

This manual describes RT-11 Version 2 software (identification numbers are usually produced via a specific command in each program, or are part of the header in the listing produced by the program; see the appropriate chapter or appendix):



## Preface

| <u>Program</u>     | <u>Version 2<br/>Identification</u> | <u>Version 1<br/>Identification</u> |
|--------------------|-------------------------------------|-------------------------------------|
| Single-Job Monitor | V02-01                              | V01-15                              |
| F/B Monitor        | V02-01                              | N/A                                 |
| EDIT               | V02-02                              | V01-24                              |
| PIP                | V02-02                              | V01-06                              |
| MACRO              | VM02-09                             | VM01-01                             |
| CREF               | V01-02                              | N/A                                 |
| LINK               | V03-01                              | V01-04                              |
| LIBR               | V02-01                              | N/A                                 |
| ODT                | V01-01                              | V01-01                              |
| EXPAND             | V02-02                              | V01                                 |
| ASEMBL             | VS02-09                             | VS01-01                             |
| FILEX              | V01-01                              | N/A                                 |
| SRCCOM             | V01-01                              | N/A                                 |
| DUMP               | V01-01                              | N/A                                 |
| PATCH              | V01-02                              | V01-01                              |
| PATCHO             | V01-02                              | N/A                                 |

RT-11 Version 2 is compatible with RT-11 Version 1 with respect to the following:

1. Version 1 source programs will assemble properly under Version 2 (the OBJ modules are compatible).
2. Version 1 save image files which conform to RT-11 programming rules (see Chapter 9 and Appendix H) will execute correctly under Version 2.

Differences between the two systems include:(1)

1. Revised device handler and interrupt interfaces have been written for Version 2 (see Appendix H).
2. Stricter rules for the .SETTOP request must be observed in Version 2 (see Chapter 9).
3. Additional monitor commands are available in Version 2, with extension of some of the pre-existing commands (see Chapter 2).

Additional features available in Version 2 and not in Version 1 include:

1. Support of RF fixed-head disk, card reader, magtape, cassette and display hardware
2. Foreground/Background Monitor (including timer support)

---

(1) See Chapter 5 in "Getting Started With RT-11" (DEC-11-ORCPA-D-D) for a comprehensive list of differences between the two systems.



## Preface

3. Extended programmed requests
4. New utilities, including CREF, LIBR, FILEX, SRCCOM, DUMP, PATCHO, and optionally, FORTRAN IV.

In general, if the user is already familiar with RT-11, Version 1, he may omit reading the following chapters, since the technical information has not changed:

- |              |         |                                    |
|--------------|---------|------------------------------------|
| Chapter 3 -  | EDIT    | (with exception of Section 3.7)    |
| Chapter 5 -  | MACRO   | (with exception of Section 5.7)    |
| Chapter 8 -  | ODT     | (note restrictions in Section 8.1) |
| Chapter 10 - | EXPAND  |                                    |
| Chapter 11 - | ASSEMBL |                                    |
| Appendix L - | PATCH   |                                    |

The remainder of the manual should be read in its entirety. The user should not assume that knowledge of Version 1 is sufficient for use of Version 2.

Three additional manuals describe the capabilities of the optional RT-11 system components:

1. BASIC/RT-11 LANGUAGE REFERENCE MANUAL (DEC-11-LBACA-C-D-D)
2. PDP-11 FORTRAN LANGUAGE REFERENCE MANUAL (DEC-11-LFLRA-A-D)
3. RT-11 FORTRAN COMPILER AND OBJECT TIME SYSTEM USER'S MANUAL (DEC-11-LRFPA-A-D)

These manuals can be ordered from the Digital Software Distribution Center. Summaries of the features provided by each language appear in this manual in Appendixes F and G respectively.

Two PDP-11 system manuals may be helpful when using FILEX (Appendix J) to convert programs between DOS, RSTS-11 and RT-11 formats:

1. PDP-11 RESOURCE SHARING TIME-SHARING SYSTEM USER'S GUIDE (DEC-11-ORSUA-A-D)
2. PDP-11 DISK OPERATING SYSTEM MONITOR PROGRAMMER'S HANDBOOK (DEC-11-OMONA-A-D)

Users of display hardware may wish to refer to the appropriate hardware manual:

1. GT40 USER'S GUIDE (DEC-11-HGTGA-A-D)
2. GT44 USER'S GUIDE (DEC-11-HGT44-A-D)
3. VT11 GRAPHIC DISPLAY PROCESSOR MANUAL (DEC-11-HVTGA-A-D)
4. VT50 VIDEO TERMINAL PROGRAMMER'S MANUAL (DEC-00-OV TSA-A-D).

Conventions used throughout this manual include the following:

1. Actual computer output is used in examples wherever possible. When necessary, computer output is underlined to differentiate from user responses.



## Preface

2. To avoid confusion, a line feed (character or key) is represented in the text as <LF>; a carriage return (character or key) is represented as <CR>. Unless otherwise indicated, all commands and command strings are terminated by a carriage return.
3. Terminal, console terminal, and teleprinter are general terms used throughout the documentation to represent any one of the following: LA30 or LA36 DECwriter, VT05 or VT50 Display, LT33 or LT35 Teletype(2).
4. Several characters in system commands are produced by typing a combination of keys concurrently; for example, the CTRL key is held down while typing an O to produce a command which causes suppression of teleprinter output. Key combinations such as this are documented as CTRL O, CTRL C, SHIFT N, and so forth.

---

(2) Teletype is a registered trademark of the Teletype Corporation.



## CHAPTER 1

### RT-11 OVERVIEW

RT-11 is a single-user programming and operating system designed for the PDP-11 series of computers. This system permits the use of a wide range of peripherals and up to 28K of either solid state or core memory (hereafter referred to as memory).

RT-11 provides two operating environments: Single-Job operation, and a powerful Foreground/Background (F/B) capability(1).

Single-Job operation allows only one program to reside in memory at any time; execution of the program continues until either it is completed or it is physically interrupted by the user at the console.

In a Foreground/Background environment, two independent programs may reside in memory. The foreground program is given priority and executes until it relinquishes control to the background program; the background program is allowed to execute until control is again required by the foreground program, and so on. This sharing of system resources greatly increases the efficiency of processor usage.

To handle both operating environments, RT-11 offers two completely compatible and versatile monitors (Single-job and F/B); either monitor provides complete user control of the system from the console terminal keyboard. Monitor commands which allow the user to direct single-job, foreground, and background operations are described in Chapter 2.

In addition to the monitor facilities, RT-11 offers a full complement of system programs; these allow program development using high level languages such as FORTRAN IV and BASIC or assembly language (MACRO or EXPAND/ASEMBL). System programs are summarized in Section 1.2 and are discussed in detail in individual chapters and appendixes of this manual.

---

(1) The uses and advantages of each environment are outlined later in this chapter.



## RT-11 Overview

### 1.1 PROGRAM DEVELOPMENT

Computer systems such as RT-11 are often used extensively for program development. The programmer makes use of the programming "tools" available on his system to develop programs which will perform functions specific to his needs. The number and type of "tools" available on any given system depend on a good many factors--the size of the system, its application and its cost, to name a few. Most Digital systems, however, provide several basic program development aids: these generally include an editor, assembler, linker, debugger, and often a librarian; a high level language (such as FORTRAN IV or BASIC) is also usually available.

An editor is used to create and modify textual material. Text may be the lines of code which make up a source program written in some programming language, or it may be data; text may be reports, or memos, or in fact may consist of any subject matter the user wishes. In this respect using an editor is analogous to using a typewriter--the user sits at a keyboard and types text. But the advantages of an editor far exceed those of a typewriter because once text has been created, it can be modified, relocated, replaced, merged, or deleted--all by means of simple editing commands. When the user is satisfied with his text, he can save it on a storage device where it is available for later reference.

If the editor is used for the purpose of writing a source program, development does not stop with the creation of this program. Since the computer cannot understand any language but machine language (which is a set of binary command codes), an intermediary program is necessary which will convert source code into the instructions the computer can execute. This is the function of an assembler.

The assembler accepts alphanumeric representations of PDP-11 coding instructions (i.e., mnemonics), interprets the code, and produces as output the appropriate object code. The user can direct the assembler to generate a listing of both the source code and binary output, as well as more specific listings which are helpful during the program debugging process. In addition, the assembler is capable of detecting certain common coding errors and of issuing appropriate warnings.

The output produced by the assembler is called object output because it is composed of object (or binary) code. On PDP-11 systems, the object output is called a module and contains the user's source program in the binary language which is acceptable to a PDP-11 computer.

Source programs may be complete and functional by themselves; however, some programs are written in such a way that they must be used in conjunction with other programs (or modules) in order to form a complete and logical flow of instructions. For this reason the object code produced by the assembler must be relocatable--that is, assignment of memory locations must be deferred until the code is combined with all other necessary object modules. It is the purpose of linker to perform this relocation.

The linker combines and relocates separately assembled object programs. The output produced by the linker consists of a load module, which is the final linked program ready for execution. The user can, at his option, request a load map which displays all addresses assigned by the linker.



## RT-11 Overview

Very rarely is a program created which does not contain at least one unintentional error, either in the logic of the program or in its coding. Errors may be discovered by the programmer while he is editing his program, or the assembler may find errors during the assembly process and inform the programmer by means of error codes. The linker may also catch certain errors and issue appropriate messages. Often, however, it is not until execution that the user discovers his program is not working properly. Programming errors may be extremely difficult to find, and for this reason a debugging tool is usually available to aid the programmer in determining the cause of his error.

A debugging program allows the user to interactively control the execution of his program. With it, he can examine the contents of individual locations, search for specific bit patterns, set designated stopping points during execution, change the contents of locations, continue execution, and test the results, all without the need of re-editing and re-assembling.

When programs are successfully written and executed, they may be useful to other programmers. Often routines which are common to many programs (such as I/O routines) or sections of code which are used over and over again, are more useful if they are placed in a library where they can be retrieved by any interested user. A librarian provides such a service by allowing creation of a library file. Once created, the library can be expanded, updated, or listed.

High level languages simplify the programmer's work by providing an alternate means of writing a source program other than assembly language mnemonics. Generally, high level languages are easy to learn--a single command may cause the computer to perform many machine language instructions. The user does not need to know about the mechanics of the computer to use a high level language. In addition, some high level languages (like BASIC) offer a special immediate mode which allows the user to solve equations and formulas as though he were using a calculator. Assembling and linking are done automatically so that the user can concentrate on solving the problem rather than using the system.

These are a few of the programming tools offered by most computer systems. The next section summarizes specific programming aids available to the user of RT-11.

### 1.2 SYSTEM SOFTWARE COMPONENTS

The following is a brief summary of the RT-11 system programs:

1. The Text Editor (EDIT, described in Chapter 3) is used to create or modify source files for use as input to language processing programs such as the assembler or FORTRAN. EDIT contains powerful text manipulation commands for quick and easy editing of a text file. EDIT also allows use of a VT-11 display processor (such as the GT44), if one is part of the hardware configuration (see Section 1.3).
2. The MACRO Assembler (Chapter 5) brings the capabilities of macros to the RT-11 system with 12K (or more) memory. (Macros are instructions in a source or command language which are equivalent to a specified sequence of machine



## RT-11 Overview

instructions or commands.) The assembler accepts source files written in the MACRO language and generates a relocatable object module to be processed by the Linker before loading and execution. Cross reference listings of assembled programs may be produced using CREF in conjunction with the MACRO Assembler.

3. EXPAND (Chapter 10) is used in an 8K F/B job area or 8K systems (or in larger systems with programs of great size) to expand macros in an assembly language program into macro-free source code, thus allowing the program to be assembled in 8K using ASEMBL.
4. ASEMBL (Chapter 11) is an assembler designed for use in an 8K RT-11 system, an 8K F/B job area, or larger systems where symbol table space is a factor. ASEMBL is a subset of MACRO-11 with more limited features. (CREF is not available under ASEMBL.)
5. The Linker (LINK, described in Chapter 6) fixes (i.e., makes absolute) the values of relocatable symbols and converts the relocatable object modules of compiled or assembled programs and subroutines into a load module which can be loaded and executed by RT-11. LINK can automatically search library files for specified modules and entry points; it can produce a load map (which lists the assigned absolute addresses) and can provide automatic overlay capabilities to very large programs. The Linker can also produce files suitable for running in the foreground.
6. The Librarian (LIBR, see Chapter 7) allows the user to create and maintain his own library of functions and routines. These routines are stored on a random access device as library files, where they can be referenced by the Linker.
7. The Peripheral Interchange Program (PIP, see Chapter 4) is the RT-11 file maintenance and utility program. It is used to transfer files between all devices which are part of the RT-11 system, to rename or delete files, and to obtain directory listings.
8. SRCCOM (Source Compare, described in Appendix K) allows the user to perform a character-by-character comparison of two or more text files. Differences can be listed in an output file or directly on the line printer or terminal, thus providing a fast method of determining, for example, if all edits to a file have been correctly made.
9. FILEX (Appendix J) allows file transfers to occur between DECTapes used under the DECsystem-10 or PDP-11 RSTS system, and DECTape and disk used under the DOS/BATCH system, and any RT-11 device.
10. The PATCH utility program (Appendix L) is used to make minor modifications to memory image files (output files produced by the Linker); it is used on files which do or do not have overlays. PATCHO (Appendix M) is used to make minor modifications to files in object format (output files produced by the FORTRAN compiler and the Librarian, or MACRO and ASEMBL assemblers).



## RT-11 Overview

11. ODT (On-line Debugging Technique, described in Chapter 8) aids in debugging assembled and linked object programs. It can print the contents of specified locations, execute all or part of the object program, single step through the object program, and search the object program for bit patterns.
12. DUMP (Appendix I) is used to print for examination all or any part of a file in octal words, octal bytes, ASCII and/or RAD50 characters (see Chapter 5).

BASIC and FORTRAN IV are two high level languages available under RT-11. Summaries of their language features and commands are provided in Appendixes F and G of this manual.

### 1.3 SYSTEM HARDWARE COMPONENTS

The minimum RT-11 system (that is, one that does not use the F/B capability) requires a PDP-11 series computer with at least 8K of memory, a random access device and a console terminal. To use the GT ON and GT OFF monitor commands, a system configuration of at least 12K is required. If the F/B capability is to be used, at least 16K of memory and a line frequency clock are required.

The following are optional hardware devices supported by the RT-11 system:

RK05/RK11, RF11/RS11 disk  
TC11/TU56 DECTape  
Display Processor (VT11)  
Line Printer (LP11, LS11, or LV11)  
Card Reader (CR11, CM11)  
Terminal (LA30/LA36, VT05, VT50 or LT33/LT35)  
PC11 High-speed Reader/Punch  
TM11/TU10 Magtape  
TA11/TU60 Cassette

RT-11 operates in environments ranging from 8K to 28K words of memory. Reconfiguration for different memory sizes is unnecessary--the same system device operates on any PDP-11 processor with 8K to 28K of memory and makes use of all memory available.



## RT-11 Overview

### 1.4 USING THE RT-11 SYSTEM

As mentioned earlier in the chapter, the RT-11 system offers two complete operating environments. Each is controlled by a single user from the console terminal keyboard by means of an appropriate monitor--Single-Job or Foreground/Background. Both monitors are completely compatible and allow full user interaction with all features which are a part of the operating environment in use.

The choice of which environment to use, and, consequently, which monitor to run, depends upon the needs of the user. The next two sections provide information useful in determining which monitor is more suitable for certain applications.

#### 1.4.1 RT-11 Single-Job Monitor

The RT-11 Single-Job Monitor provides a single-user, single-program system which can operate in as little as 8K of memory. Since the Single-Job Monitor itself requires approximately one-half the memory space needed by the Foreground/Background Monitor, this system is ideal for extensive program development work; a much larger area of memory is available for the user program and its buffers and tables. Programs requiring extremely high data rates are best run in the Single-Job environment, since interrupts can be serviced at a much higher rate.

All system programs (listed in Section 1.2) can be used under the Single-Job Monitor, and many of the features of the Foreground/Background Monitor (i.e., KMON commands and programmed requests not used to control foreground jobs) are supported.

In effect, the Single-Job Monitor is much smaller and slightly faster than the Foreground/Background Monitor; it can best be used when program size is the important factor.

#### 1.4.2 RT-11 Foreground/Background Monitor

Quite often the central processor of a computer system may spend a large percentage of time waiting for some external event to occur, the most common event being the completion of an I/O transfer (this is particularly true of real time jobs). Many users would like to take advantage of this unused capacity to accomplish other lower-priority tasks such as further program development or complex data analysis. The Foreground/Background system provides this capability.

In a Foreground/Background system the foreground job is the time-critical, on-line job, and is given top priority; whenever possible the processor runs the foreground job. However, when the foreground job reaches a state in which no more processing can be done until some external event occurs, the monitor will try to run the lower priority background job. The background job then runs until the foreground job is again in a runnable state, at which point the processor will interrupt the background job and resume the foreground job.

In general, the RT-11 Foreground/Background System is designed to allow a time-critical job to run in the foreground, while the



## RT-11 Overview

background does non-time-critical jobs, such as program development. (All RT-11 system programs run as the background job in a F/B system.) Thus, the user can run FORTRAN, BASIC, MACRO, etc. in the background while the foreground may be collecting data and storing and/or analyzing it.

Most user programs written for an RT-11 System can be linked (using the Linker described in Chapter 6) to run as the foreground job. There are a few coding restrictions, and these are explained in Appendix H, F/B Programming and Device Handlers. A foreground program has access to all of the features available to the background job (opening and closing files, reading and writing data, etc.). In addition, the F/B System gives the user the ability to set timer routines, suspend and resume F/B jobs, and send data and messages between the two jobs.

### 1.4.3 Facilities Available Only in RT-11 F/B

As mentioned previously, RT-11 F/B allows the user to write and execute two independent programs. Some features which are available only to the F/B user include:

1. Mark Time--This facility allows user programs to set clock timers to run for specified amounts of time. When the timer runs out, a routine specified by the user is entered. There may be as many mark time requests as desired, providing system queue space is reserved (see .QSET, Chapter 9).
2. Timed Wait--This feature allows the user program to "sleep" until the specified time increment elapses. Typically, a program may need to sample data every few seconds or even minutes. While the program is idle, the other job can run. The timed wait accomplishes this; when the time has elapsed, the issuing job is again runnable (see .TWAIT, Chapter 9).
3. Send Data/Receive Data--It is possible, under RT-11 F/B, to have the foreground and background programs communicate with one another. This is accomplished with the send/receive data functions. Using this facility, one program sends messages (or data) in variable size blocks to the other job. This can be used, for example, to pass data from a foreground collection program directly to a background analysis program (see .SDAT/.RCVD, Chapter 9).



Dear Sir,

I have the honor to acknowledge the receipt of your letter of the 10th inst. in relation to the above matter.

I am sorry to hear that you are not satisfied with the results of the examination. I have been very anxious to see that the work was done to the best of my ability, and I am sure that the results are as good as can be expected under the circumstances.

I have been very careful to see that the work was done to the best of my ability, and I am sure that the results are as good as can be expected under the circumstances.

I am sorry to hear that you are not satisfied with the results of the examination. I have been very anxious to see that the work was done to the best of my ability, and I am sure that the results are as good as can be expected under the circumstances.

I have been very careful to see that the work was done to the best of my ability, and I am sure that the results are as good as can be expected under the circumstances.

I am sorry to hear that you are not satisfied with the results of the examination. I have been very anxious to see that the work was done to the best of my ability, and I am sure that the results are as good as can be expected under the circumstances.

I have been very careful to see that the work was done to the best of my ability, and I am sure that the results are as good as can be expected under the circumstances.

I am sorry to hear that you are not satisfied with the results of the examination. I have been very anxious to see that the work was done to the best of my ability, and I am sure that the results are as good as can be expected under the circumstances.

I have been very careful to see that the work was done to the best of my ability, and I am sure that the results are as good as can be expected under the circumstances.



## CHAPTER 2

### SYSTEM COMMUNICATION

The monitor is the hub of RT-11 system communications; it provides access to system and user programs, performs input and output functions, and enables control of background and foreground jobs.

The user communicates with the monitor through programmed requests and keyboard commands. The keyboard commands (described in Section 2.7) are used to load and run programs, start or restart programs at specific addresses, modify the contents of memory, and assign and deassign alternate device names.

Programmed requests (described in detail in Chapter 9) are source program instructions which pass arguments to the monitor and request monitor services. These instructions allow user assembly language programs to utilize the available monitor features.

#### 2.1 START PROCEDURE

After the system has been built (see Appendix A and the GETTING STARTED WITH RT-11 document, DEC-11-ORCPA-D-D), the monitor can be loaded into memory from disk or DECTape as follows:

1. Press HALT.
2. Mount the system device on unit 0. Note that if disk unit 0 is disabled, RT-11 can be loaded from another RK disk unit using the software bootstrap described later in this section.
3. If a disk is being used, be sure the WRITE PROTECT light is lit.
4. If a DECTape unit is being used, set the WRITE ENABLE/WRITE LOCK switch to WRITE LOCK and the REMOTE/OFF/LOCAL switch to REMOTE.

If the hardware configuration includes the BM792-YB hardware bootstrap:

1. Set the Switch Register to 173100 (the address of the ROM Bootstrap Loader).



# System Communication

2. Press the LOAD ADDR switch.
3. Set the Switch Register to the address of the word count register of disk or DECTape on which the monitor resides:

|        |                     |
|--------|---------------------|
| 177462 | for RF11 disk       |
| 177406 | for RK11, RK05 disk |
| 177344 | for DECTape         |

4. Press the START switch.

If the hardware configuration includes the MR11-DB hardware bootstrap:

1. Set the Switch Register to:

|        |                     |
|--------|---------------------|
| 773100 | for RF11 disk       |
| 773110 | for RK11, RK05 disk |
| 773120 | for DECTape         |

2. Press the LOAD ADDR switch.

3. Press the START switch.

If neither hardware bootstrap is available, or if an RK disk unit other than 0 is to be used as the system device, one of the following bootstraps must be entered manually using the switch register. First set the switch register to 1000 and press the LOAD ADDR switch. Then set the switch register to the first value shown for the appropriate bootstrap and raise the DEPOSIT switch. Continue depositing the values shown.

| <u>DECTape</u> | <u>Disk<br/>(RK11, RK05)</u> | <u>RK Disk (other<br/>than Unit 0)</u> | <u>Disk<br/>(RF11)</u> |
|----------------|------------------------------|--|------------------------|
| 12700          | 12700                        | 12700                                  | 12700                  |
| 177344         | 177406                       | 177406                                 | 177466                 |
| 12710          | 12710                        | 12760                                  | 5010                   |
| 177400         | 177400                       | xxxxxxx *                              | 5040                   |
| 12740          | 12740                        | 4                                      | 12740                  |
| 4002           | 5                            | 12700                                  | 177400                 |
| 5710           | 105710                       | 177406                                 | 12740                  |
| 100376         | 100376                       | 12710                                  | 5                      |
| 12710          | 5007                         | 177400                                 | 105710                 |
| 3              |                              | 12740                                  | 100376                 |
| 105710         |                              | 5                                      | 5007                   |
| 100376         |                              | 105710                                 |                        |
| 12710          |                              | 100376                                 |                        |
| 5              |                              | 5007                                   |                        |
| 105710         |                              |  |                        |
| 100376         |                              |  |                        |
| 5007           |                              |  |                        |

|            |                   |
|------------|-------------------|
| * xxxxxx = | 20000 for unit 1  |
|            | 40000 for unit 2  |
|            | 60000 for unit 3  |
|            | 100000 for unit 4 |
|            | 120000 for unit 5 |
|            | 140000 for unit 6 |
|            | 160000 for unit 7 |



## System Communication

When all the values have been entered, set the switches to 1000 and press the LOAD ADDR and START switches.

The monitor loads into memory and prints one of the following identification messages followed by a dot (.) on the terminal:

```
RT-11SJ V02-01  
RT-11FB V02-01
```

The message printed indicates which monitor (Single-Job or F/B) has been loaded; the user may determine which is to be loaded during the system build operation. Refer to the building instructions in Appendix A or the GETTING STARTED WITH RT-11 document for instructions on how to initialize the system for either monitor and for directions on how to bring up the alternate monitor while under control of the one currently running.

After the message has printed, the system device should be WRITE ENABLED. The monitor is ready to accept keyboard commands.

## 2.2. SYSTEM CONVENTIONS

Special character commands, file naming procedures and other conventions that are standard for the RT-11 system are described in this section. The user should be familiar with these conventions before running the system.

### 2.2.1 Data Formats

The RT-11 system makes use of five types of data formats: ASCII, object, memory image, relocatable image, and load image.

Files in ASCII format conform to the American National Standard Code for Information Interchange, in which each character is represented by a 7-bit code. Files in ASCII format include program source files created by the Editor, listing and map files created by various system programs, and data files consisting of alphanumeric characters. A chart containing ASCII character codes appears in Appendix C.

Files in object format consist of data and PDP-11 machine language code. Object files are those output by the assembler or FORTRAN compiler and are used as input to the Linker.

The Linker can output files in memory image format (.SAV), relocatable image format (.REL), or load image format (.LDA).

A memory image file (.SAV) is a 'picture' of what memory will look like when a program is loaded. The file itself requires the same number of disk blocks as the corresponding number of 256-word memory blocks.

A relocatable image file (.REL) is one which can be run in the foreground. It differs from a memory image file in that the file is linked as though its bottom address were 0. When the program is called (using the monitor FRUN command), the file is relocated as it is loaded into memory. (A memory image file requires no such relocation.)



## System Communication

A load image (or .LDA) file may be produced for compatibility with the PDP-11 Paper Tape System and is loaded by the absolute binary loader. LDA files can be loaded and executed in stand-alone environments without relocation.

### 2.2.2 Prompting Characters

The following table summarizes the characters typed by RT-11 to indicate to the user either that the system is awaiting user response or to specify which job (foreground or background) is producing output:

Table 2-1  
Prompting Characters

| Character | Meaning  |
|-----------|--|
| .         | The Keyboard Monitor is waiting for a command (see Section 2.3.2).   |
| *         | The Command String Interpreter is waiting for a command string specification as explained in Sections 2.3.3 and 2.5.   |
| ↑         | When the console terminal is being used as an input file, the uparrow prompts the user to enter information from the keyboard. If the input is entered under EDIT or BASIC (or any program that accepts input in special terminal mode as described in Chapter 9), the characters entered are not echoed. Typing a CTRL Z marks the end-of-file.   |
| >         | The > character is used (under the F/B Monitor and only if a foreground job is active) to identify which job, foreground or background, is producing the output currently appearing on the console terminal. Each time output from the background job is to appear, B> is printed first, followed by the output. If the foreground job is to print output, F> is typed first. B> and F> are also printed as a result of the CTRL B and CTRL F commands described in Table 2-4. |

### 2.2.3 Physical Device Names

Devices are referenced by means of a standard two-character device name. Table 2-2 lists each name and its related device. If no unit number is specified for devices which have more than one unit, unit 0 is assumed.



## System Communication

Table 2-2  
Permanent Device Names

| Permanent Name | I/O Device  |
|----------------|---|
| CR:            | Card Reader (CR11/CM11).  |
| CTn:           | TAll cassette (n is the unit number, 0 or 1)  |
| DK:            | The default storage device for all files. DK is initially the same as SY: (see below), but the assignment can be changed with the ASSIGN Command (Section 2.7.2.4.) |
| DTn:           | DEctape n, where n is a unit number (an integer in the range 0 to 7, inclusive).  |
| LP:            | Line printer.   |
| MTn:           | Industry compatible magtape (n is an integer between 0 and 7, inclusive).   |
| PP:            | High-speed paper tape punch.  |
| PR:            | High-speed paper tape reader.   |
| RF:            | RF11 fixed-head disk drive.   |
| RKn:           | RK disk cartridge drive n (n is in the range 0 to 7 inclusive).   |
| SY:            | System device; the device and unit from which the system is bootstrapped. (RT-11 allows bootstrapping from any RK unit; refer to Section 2.1.)                      |
| SYn:           | The specified unit of the same device type as that from which the system was bootstrapped.  |
| TT:            | Terminal keyboard and printer.  |

In addition to the fixed names shown in Table 2-2, devices can be assigned logical names. A logical name takes precedence over a physical name and thus provides device independence. With this feature a program that is coded to use a specific device does not need to be rewritten if the device is unavailable. Refer to Section 2.7.2.4 for instructions on assigning logical names to devices.

### 2.2.4 File Names and Extensions

Files are referenced symbolically by a name of one to six alphanumeric characters followed, optionally, by a period and an extension of up to three alphanumeric characters. (Excess characters in a filename may cause an error message.) The extension to a filename generally indicates the format of a file. It is a good practice to conform to



## System Communication

the standard filename extensions for RT-11. If an extension is not specified for an input or output file, most system programs assign appropriate default extensions. Table 2-3 lists the standard extensions used in RT-11.

Table 2-3  
File Name Extensions

| Extension | Meaning   |
|-----------|---|
| .BAD      | Files with bad (unreadable) blocks; this extension can be assigned by the user whenever bad areas occur on a device. The .BAD extension makes the file permanent in that area, preventing other files from using it and consequently becoming unreadable. |
| .BAK      | Editor backup file.   |
| .BAS      | BASIC source file (BASIC input).  |
| .DAT      | BASIC or FORTRAN data file.   |
| .DMP      | DUMP output file.   |
| .FOR      | FORTRAN IV source file (FORTRAN input).   |
| .LDA      | Absolute binary file (optional Linker output).  |
| .LLD      | Library listing file.   |
| .LST      | Listing file (MACRO or FORTRAN output).   |
| .MAC      | MACRO or EXPAND source file (MACRO, EXPAND, SRCCOM input).  |
| .MAP      | Map file (Linker output).   |
| .OBJ      | Relocatable binary file (MACRO, ASEMBL, FORTRAN IV output, Linker input, LIBR input and output).  |
| .PAL      | Output file of EXPAND (the MACRO expander program), input file of ASEMBL.   |
| .REL      | Foreground job relocatable image (Linker output, default for monitor FRUN command).   |
| .SAV      | Memory image or SAVE file; default for R, RUN, SAVE and GET Keyboard Monitor commands; also default for output of Linker.   |
| .SYS      | System files and handlers.  |



## System Communication

If a filename with a blank extension is to be used in a command line in which a default extension is assumed (by either the monitor or a system program), the user must insert a period after the filename to indicate that there is no extension. For example, to run the file TEST, type:

RUN TEST.

If the period after the filename is not given, the monitor assumes the .SAV extension and attempts to run a file named TEST.SAV.

### 2.3 MONITOR SOFTWARE COMPONENTS

The main RT-11 monitor software components are:

Resident Monitor (RMON)

Keyboard Monitor (KMON)

User Service Routine (USR) and Command String Interpreter (CSI)

Device Handlers

The reader may find Figure 2-1 helpful while reading the following descriptions.

#### 2.3.1 Resident Monitor (RMON)

The Resident Monitor is the only permanently memory-resident part of RT-11. The programmed requests for all services of RT-11 are handled by RMON. RMON also contains the console terminal service, error processor, system device handler, EMT processor, and system tables.

#### 2.3.2 Keyboard Monitor (KMON)

The Keyboard Monitor provides communication between the user at the console and the RT-11 system. Monitor commands allow the user to assign logical names to devices, run programs, load device handlers, and control F/B operations. A dot at the left margin of the console terminal page indicates that the Keyboard Monitor is in memory and is waiting for a user command.

#### 2.3.3 User Service Routine (USR)

The User Service Routine provides support for the RT-11 file structure. It loads device handlers, opens files for read or write operations, deletes and renames files, and creates new files. The Command String Interpreter (the use of which is described in Section 2.5) is part of the USR and can be accessed by any program to interpret device and file I/O information.



## System Communication

### 2.3.4 Device Handlers

Device handlers for the RT-11 system perform the actual transfer of data to and from peripheral devices. New handlers can be added to the system as files on the system device and can be interfaced to the system by modifying a few monitor tables (see the RT-11 SOFTWARE SUPPORT MANUAL, DEC-11-ORPGA-B-D for instructions on how to interface a new handler to the RT-11 monitor).

### 2.4 GENERAL MEMORY LAYOUT

When the RT-11 System is first bootstrapped from the system device, memory is arranged as shown in the left diagram of Figure 2-1 (this is the case for either the Single-Job or Foreground/Background Monitor, since no foreground job exists yet). The background job is the RT-11 module KMON.

When an RT-11 foreground job is initiated (via the monitor FRUN command, Section 2.7.5.1), room is created for the foreground job to be loaded by decreasing the amount of space available to the background job. The memory maps in Figure 2-1 illustrate the system layout before and after a foreground job is loaded. (Refer also to Chapter 6, Section 6.5.)

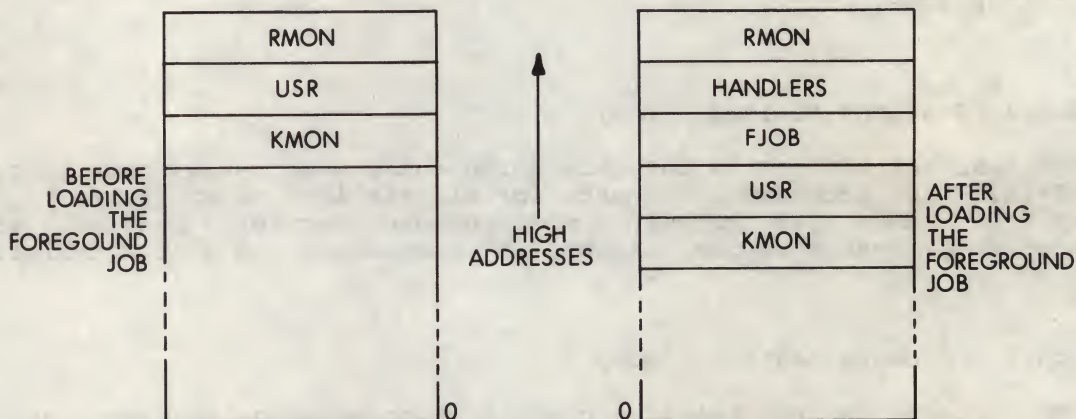


Figure 2-1  
RT-11 System Memory Maps

As shown in the figures, the process of loading a foreground job requires that the USR and KMON be physically moved. Once a foreground job is running, it is possible to communicate with either the background or foreground job via special commands (described in Section 2.7). All of the terminal support functions described in Section 2.6 are available under both the Single-job and F/B Monitors.

In addition to FRUN, other monitor commands can alter the memory map; these are LOAD, UNLOAD, GT ON, and GT OFF. LOAD causes device handlers to be made resident until an UNLOAD command is performed. UNLOAD deletes handlers which have been loaded. GT ON and GT OFF cause terminal service to utilize the VT-11 display hardware. Figure 2-2 illustrates the placement of display modules and device handlers in memory following the GT ON, LOAD, and FRUN commands:



## System Communication

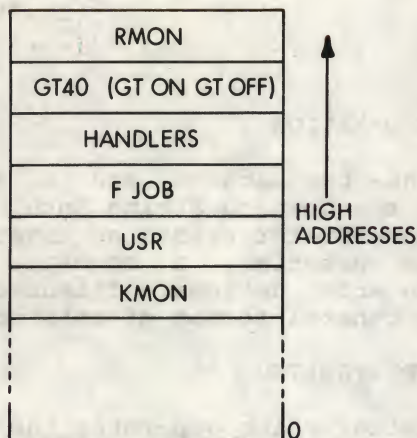


Figure 2-2  
RT-11 Memory Map (GT40)

RT-11 maintains a free memory list to manage memory. Thus, when a handler is unloaded, the space the handler occupied is returned to the free memory list and is reclaimed by the background.

### 2.4.1 Component Sizes

Following are the approximate sizes (in words) of the components for RT-11, Version 2.

|      | <u>F/B</u> | <u>Single-job</u> |
|------|------------|-------------------|
| RMON | 3400 (10)  | 1650 (10)         |
| USR  | 2050 (10)  | 2050 (10)         |
| KMON | 1550 (10)  | 1550 (10)         |

In the F/B system, the background area must always be large enough to hold KMON and USR (3.5K words). The following list indicates the total space available for the loaded device handlers, the foreground job, and the display handler. Note that the low memory area from 0-477 is never used for executable programs. (These sizes also allow room for the 3.5K RMON).

| <u>Machine size (words)</u> | <u>Space available (words)</u> |
|-----------------------------|--------------------------------|
| 16K                         | 9K                             |
| 24K                         | 17K                            |
| 28K                         | 21K                            |

With the Single-Job Monitor, RMON requires only 1.65K. The following list shows the amount of space available to users with the Single-Job Monitor:



## System Communication

| <u>Machine size (words)</u> | <u>Program space available (words)</u> |
|-----------------------------|--|
| 8K                          | 6.35K                                  |
| 16K                         | 14.35K                                 |
| 24K                         | 22.35K                                 |
| 28K                         | 26.35K                                 |

### 2.5 ENTERING I/O INFORMATION

Once either monitor has been loaded and a system program (or any program which uses the Command String Interpreter) has been started, the Command String Interpreter prints an asterisk at the left margin. In response to the asterisk, a command string should be entered providing information about devices, filenames and extensions, and switch options. The general format of this command line is:

`*OUTPUT=INPUT/SWITCH`

The = sign is a delimiter which separates the output and input fields; the < sign may be used in place of the = sign. See the NOTE at the end of this section.

OUTPUT is entered in the format:

`dev:filnam.ext[n],...dev:filnam.ext[n]`

INPUT as:

`dev:filnam.ext,...dev:filnam.ext`

and SWITCH as:

`/s:oval or /s!dval`

where:

|                    |   |
|--------------------|---|
| dev:               | in each case is an optional two to three-character name from Table 2-2 whose usage conforms to the NOTE below.  |
| filnam.ext         | in each case is the name of a file (consisting of one to six alphanumeric characters followed optionally by a dot and a zero to three-character extension). As many as three output and six input files may be allowed.   |
| [n]                | is an optional declaration of the number of blocks (n) desired for an output file. n is a decimal number (<2(16)-1) enclosed in square brackets immediately following the output filnam.ext to which it applies.  |
| /s:oval or /s!dval | is one or more optional switches whose functions vary according to the program in use (refer to the switch option table in the appropriate chapter). oval is either an octal number or one to three alphanumeric characters (the first of which must be alphabetic) which will be converted to radix-50 (see Section 5.5.4 of the MACRO chapter). dval is a decimal value preceded by an exclamation point. |



## System Communication

Throughout this manual, the /s:oval construction is used; however, the /s!dval format is always valid. Generally, these switches and their associated values, if any, should follow the device and filename to which they apply.

If the same switch is to be repeated several times with different values (e.g., /L:MEB/L:TTM/L:CND to MACRO) the line may be abbreviated as /L:MEB:TTM:CND; octal, RAD50, and decimal values may be mixed.

### NOTE

As illustrated in the general format of a command line, the command line consists of an output list, a separator (= or <), and an input list. The separator can be omitted if there are no output files. Omission of a device specification in either the input or output list is handled as follows:

DK: is assumed if the first file in a list has no explicit device. DK (or the device associated with the first file) is default until another device is indicated; that device then becomes default until a new one is used, and so on. If the following command is entered, for example, to MACRO:

```
*DT1:FIRST.OBJ,LP:=TASK.1,RK1:TASK.2,TASK.3
```

it is interpreted as though all devices had been indicated as follows:

```
*DT1:FIRST.OBJ,LP:=DK:TASK.1,RK1:TASK.2,RK1:TASK.3
```

## 2.6 KEYBOARD COMMUNICATION (KMON)

Special function keys and keyboard commands allow the user to communicate with the RT-11 monitor and allow him to allocate system resources, manipulate memory images, start programs, and use foreground/background services.

The special functions of certain terminal keys used for communication with the Keyboard Monitor are explained in Table 2-4. Note that in the F/B system, the Keyboard Monitor always runs as a background job.

CTRL commands are entered by holding the CTRL key down while typing the appropriate letter.



# System Communication

Table 2-4  
Special Function Keys

| Key    | Function   |
|--------|--|
| CTRL A | Valid when the monitor GT ON command has been typed and the display is in use. The command does not echo on the terminal. It is used after a CTRL S has been typed to effectively page output. Console output is permitted to resume until the screen is completely filled; text previously displayed is scrolled upward off the screen. CTRL A has no special meaning if GT ON is not in effect or if a SET TTY NOPAGE command has been given (see Section 2.7.2.8).  |
| CTRL B | Under the F/B Monitor echoes B> on the terminal (unless output is already coming from the background job) and causes all keyboard input to be directed to the background job. At least one line of output will be taken from the background job (the foreground job has priority, and control will revert to it if it has output). All typed input will be directed to the background job until control is redirected to the foreground job (via CTRL F). CTRL B has no special meaning when used under a Single-Job Monitor or when a SET TTY NOFB command has been issued (see Section 2.7.2.8).   |
| CTRL C | Echoes ↑C on the terminal, interrupts current program execution, and returns control to the Keyboard Monitor. Note that under the F/B Monitor, the job which is currently receiving input will be the job that is stopped (determined by whether a CTRL F or CTRL B was most recently typed). To ensure that the command is directed to the proper job, type CTRL B or CTRL F before typing CTRL C. If a program is waiting for terminal input or is using the device handler TT for input, typing a single CTRL C interrupts execution and returns control to the monitor command level; otherwise, two CTRL C's must be typed in order to interrupt execution. |
| CTRL E | Valid when the monitor GT ON command has been typed and the display is in use. The command does not echo on the terminal, but causes all terminal output to appear on both the display screen and the console terminal simultaneously. A second CTRL E disables console terminal output. CTRL E has no special meaning if GT ON is not in effect.  |
| CTRL F | Under the F/B Monitor echoes F> on the terminal and instructs that all keyboard input be directed to the foreground job and all output be taken from the foreground job. If no foreground job exists, F? is printed and control is directed to the background job. Otherwise, control remains with the foreground job until redirected to the background job (via CTRL B) or until the foreground job terminates. CTRL F has no special meaning when used under a Single-Job Monitor, or when a SET TTY NOFB command has been used (see Section 2.7.2.8).  |



Table 2-4 (Cont.)  
Special Function Keys

| Key    | Function   |
|--------|--|
| CTRL O | <p>Echoes ↑O on the terminal and causes suppression of teleprinter output while continuing program execution. Teleprinter output is re-enabled when one of the following occurs:</p> <ol style="list-style-type: none"> <li>1. A second CTRL O is typed.</li> <li>2. A return to the monitor occurs, or</li> <li>3. The running program issues a .RCTRL O directive (see Chapter 9). (RT-11 system programs reset CTRL O to the echoing state each time a new command string is entered.)</li> </ol>               |
| CTRL Q | Does not echo. Resumes printing characters on the terminal from the point at which printing was previously stopped (via CTRL S). CTRL Q has no special meaning if a SET TTY NOPAGE command has been used (see Section 2.7.2.8).  |
| CTRL S | Does not echo. Temporarily suspends output to the terminal until a CTRL Q is typed. If GT ON is in effect, each subsequent CTRL A causes output to proceed until the screen has been refilled once. This feature allows users with high-speed terminals to fill the display screen, stop output with CTRL S, read the screen, and then continue with CTRL Q or CTRL A. (Typing CTRL C in this case also continues output.) Under the F/B Monitor, CTRL S has no special meaning if a SET TTY NOPAGE has been used. |
| CTRL U | Deletes the current input line and echoes as ↑U followed by a carriage return at the terminal. (The current line is defined to be all characters back to, but not including, the most recent line feed, CTRL C or CTRL Z.)   |
| CTRL Z | Echoes ↑Z on the terminal and terminates input when used with the terminal device handler (TT). The CTRL Z itself does not appear in the input buffer. If TT is not being used, CTRL Z has no special meaning.   |
| RUBOUT | Deletes the last character from the current line and echoes a backslash plus the character deleted. Each succeeding RUBOUT deletes and echoes another character. An enclosing backslash is printed when a key other than RUBOUT is typed. This erasure is done right to left up to the beginning of the current line.  |

### 2.6.1 Foreground/Background Terminal I/O

It is important to note that console input and output under F/B are independent functions; input can be typed to one job while output is printed by another. The user may be in the process of typing input to one job when the other job is ready to print on the terminal, in which case the job which has output interrupts the user and prints the output on the terminal; input control is not redirected to this job, however, unless a CTRL B or CTRL F is explicitly typed. If input is typed to one job while the other has output control, echo of the input



## System Communication

is suppressed until the job accepting input gains output control; at this point all accumulated input is echoed.

If the foreground job and background job are both ready to print output at the same time, the foreground job has priority. Output from the foreground job prints until a line feed is encountered, at which point output from the background job prints until a line feed is encountered, and so forth.

When the foreground job terminates, control reverts automatically to the background job.

### 2.6.2 Type-Ahead

The monitor has a type-ahead feature which allows terminal input to be entered while a program is executing. For example:

```
. R PIP
*DT1:TAPE=PR:
DT1:/L
*13-FEB-74
TAPE          78 13-FEB-74
486 FREE BLOCKS
```

While the first command line is executing, the second line (DT1:/L) is entered by the user. This terminal input is stored in a buffer and used when the first operation has completed.

If a single CTRL C is typed while in this mode, it is put into the buffer. The program currently executing exits when a terminal input request needs to be satisfied. A double CTRL C returns control to the monitor immediately.

If type-ahead input exceeds 80 characters, the terminal bell rings and no characters are accepted until part of the type-ahead buffer is used by a program or characters are deleted. No input is lost. Type-ahead is particularly useful in specifying multiple command lines to system programs, as shown in the preceding example. If a job is terminated by typing two CTRL C's, any unprocessed type-ahead is discarded.

#### NOTE

If type-ahead is used in conjunction with EDIT or BASIC, there is no terminal echo of the characters but they are stored in the buffer until a new command is needed. The characters are echoed only when actually used by the program.

## 2.7 KEYBOARD COMMANDS

Keyboard commands allow the user to communicate with the monitor. Keyboard commands can be abbreviated; optional characters in a command are delimited (in this section only) by braces. Keyboard commands require at least one space between the command and the first argument. All command lines are terminated by a carriage return.



## System Communication

All commands, with the exception of those described in Section 2.7.5, may be used under either the Single-Job or F/B Monitor. The commands described in Section 2.7.5 apply only to the F/B Monitor.

### NOTE

Any reference made to "the background job" applies as well to the Single-Job Monitor, since the background job in a F/B system is equivalent to the single-job environment in its normal state.

#### 2.7.1 Commands to Control Terminal I/O (GT ON and GT OFF)

GT ON/GT OFF

The GT ON and GT OFF commands are used to enable and disable the scroller (VT-11 display hardware). GT ON causes the display screen to replace the console as the terminal output device. Switch options allow the user to control the number of lines to appear on the screen and to position the first line vertically. Output appears on the display in the same format as it would on the console (i.e., output, text, and commands are displayed in the order in which they occur). GT ON is not permitted in an 8K configuration.

The form of the GT ON command is:

GT ON{ /L:n }{ /T:n }

where:

/L:n represents an optional switch setting indicating the number of lines of text to display; the suggested range is:

|                              |                             |
|------------------------------|-----------------------------|
| 12" screen<br>(GT40, DEClab) | 1<=n<=37 octal (31 decimal) |
|------------------------------|-----------------------------|

|                      |                             |
|----------------------|-----------------------------|
| 17" screen<br>(GT44) | 1<=n<=50 octal (40 decimal) |
|----------------------|-----------------------------|

/T:n represents an optional switch setting indicating the top position of the scroll display; the suggested range is:

|                              |                                |
|------------------------------|--------------------------------|
| 12" screen<br>(GT40, DEClab) | 1<=n<=1300 octal (744 decimal) |
|------------------------------|--------------------------------|



## System Communication

|                      |                                    |
|----------------------|------------------------------------|
| 17" screen<br>(GT44) | 1<=n<=1750 octal (1000<br>decimal) |
|----------------------|------------------------------------|

If no switches are specified, a test for the screen size is performed and default values are automatically assigned as follows:

|                              |  |
|------------------------------|--|
| 12" screen<br>(GT40, DEClab) | /L:37 (31 decimal)<br>/T:1350 (744 decimal)  |
| 17" screen<br>(GT44)         | /L:50 (40 decimal)<br>/T:1750 (1000 decimal) |

Once the display has been activated with the GT ON command, CTRL A, CTRL S, CTRL E and CTRL Q can be used to control scrolling behavior. These commands are described in Section 2.6.

### NOTE

ODT is one exception to the use of GT ON. This system program has its own terminal handler and cannot make use of the display; output will appear only on the console terminal whenever ODT is running.

The GT OFF command clears the display and resumes output on the teleprinter. The command format is:

GT OFF

If GT ON and GT OFF are used when no display hardware exists or when a foreground job is active, the ?ILL CMD? message is printed.

## 2.7.2 Commands to Allocate System Resources

DATE

2.7.2.1 DATE Command - The DATE command enters the indicated date to the system. This date is then assigned to newly created files, new device directory entries (which may be listed with PIP), and listing output until a new DATE command is issued.

The form of the command is:

DAT{E} {dd-mmm-yy}

where dd-mmm-yy is the day, month and year to be entered. dd is a decimal number in the range 1-31; mmm is the first three characters of the name of the month, and yy is a decimal number in the range 73-99. If no argument is given, the current date is printed.



## System Communication

### Examples:

DATE 21-FEB-74

Enter the date 21-FEB-74 as the current system date.

.DAT  
21-FEB-74

Print the current date.

If the date is entered in an incorrect format, the ?DAT? error message is printed.

## TIME

2.7.2.2 TIME Command - The TIME command allows the user to find out the current time of day kept by RT-11 or to enter a new time of day. If no KW11-L clock is present on the system, the ?NO CLOCK? error message is generated. If the time is entered in an incorrect format, the ?TIM? message is printed.

The form of the command is:

TIM{E} {hh:mm:ss}

where hh:mm:ss represents the hour, minute, and second. Time is represented as hours, minutes, and seconds past midnight in 24-hour format (e.g., 1:25:00 P.M. is entered as 13:25:00). If any of the arguments are omitted, 0 is assumed. If no argument is given, the current time of day is output.

### Examples:

TIM 8:15:23

Sets the time of day to 8 hours, 15 minutes and 23 seconds.

.TIM  
08:25:27

Approximately 10 minutes later, the TIME command outputs this time.

.TIME 18:5

Sets the time of day to 18:05:00.

The time of day and date are not automatically reset when the time reaches 24:00.



## INITIALIZE

2.7.2.3 INITIALIZE Command - The INITIALIZE command is used to reset several background system tables and do a general "clean-up" of the background area; it has no effect on the foreground job. In particular, this command:

1. Makes non-resident those handlers which were not loaded (via LOAD), and
2. Purges the background's I/O channels.

INITIALIZE halts any I/O currently in progress. Under the Single-Job Monitor a RESET instruction is done (see Chapter 9). Under the F/B Monitor, I/O is stopped by entering each busy device handler at a special abort entry point.

The form of the command is:

IN {INITIALIZE}

The INITIALIZE command can be used prior to running a user program, or when the accumulated results of previously issued GET commands (see Section 2.7.3.1) are to be discarded.

Example:

IN                      Initializes system background job

## ASSIGN

2.7.2.4 ASSIGN Command - The ASSIGN command assigns a user-defined (logical) name as an alternate name for a physical device. This is especially useful when a program refers to a device which is not available on a certain system. Using the ASSIGN command, I/O can be redirected to a device which is available. Only one logical name can be assigned per ASSIGN command, but several ASSIGN commands (14 maximum) can be used to assign different names to the same device. This command is also used to assign FORTRAN logical units to device names.



## System Communication

The form of the command is:

ASS{IGN} {dev} { :udev }

where:

|      |  |
|------|--|
| dev  | is any standard RT-11 (physical) device name (refer to Table 2-1) with the exception of DK and SY.   |
| udev | is a 1-3 character alphanumeric (logical) name to be used in a program to represent dev (if more than three characters are given, only the first three are actually used). |
| :    | is a delimiter character (can be a colon, equal sign, and, if separating physical and logical devices, space).   |

The placement of the delimiter is very important in the ASSIGN command; it must be placed exactly as shown in the following examples:

|                  |  |
|------------------|--|
| . ASSIGN DT1 INP | Physical device DT1 is assigned the logical device name INP. Whenever a reference to INP: is encountered, device DT1: is used. |
| . ASSIGN DT3:DK  | Physical device name DT3 is assigned the default device name DK. Whenever DK is referenced or defaulted to, DT3 is used.       |
| . ASSIGN LP=9    | FORTTRAN logical unit 9 becomes the physical device name LP. All references to unit 9 use the line printer for output.         |

Assignment of logical names to logical names is not allowed.

If only a logical device name is indicated in the command line, that particular assignment (only) is removed. Thus:

|              |  |
|--------------|--|
| . ASSIGN :9  | Deassigns the logical name 9 from its physical device (LP, in the case above).           |
| . ASSIGN =DK | Removes assignment of logical name DK from its physical device (DT3, in the case above). |

To remove all logical assignments for all physical units of a device, the following command formats are used:

|             |  |
|-------------|--|
| . ASSIGN:DT | All previous logical device assignments for all units of DECTapes are removed. |
| . ASSIGN=DT |  |

If neither a physical device name nor a logical device name is indicated, all assignments to all devices are removed.

|        |  |
|--------|--|
| ASSIGN | All previous logical device assignments are removed. |
|--------|--|



## System Communication

### CLOSE

2.7.2.5 CLOSE Command - The CLOSE command causes all currently open output files in the background job to become permanent files. If a tentative open file is not made permanent, it will eventually be deleted. The CLOSE command is most often used after CTRL C has been typed to abort a background job and to preserve any new files that job had open prior to the CTRL C; it has no effect on a foreground job.

The form of the command is:

CL{OSE}

The CLOSE command makes temporary directory entries permanent.

Example:

```
. R EDIT
*WTEXT$$
*IABCD$$
*^C
```

The Editor has a temporary file open (TEXT), which is preserved by .CLOSE.

```
. CLOSE
```

### LOAD

2.7.2.6 LOAD Command - The LOAD command is used to make a device handler resident for use with background and foreground jobs. Execution is faster when a handler is resident, although memory area for the handler must be allocated. Any device handler to be used by a foreground job must be loaded before it can be used.

The form of the command is:

LOA{D} dev{,dev=B}{,dev=F,...}

where:

|     |  |
|-----|--|
| dev | represents any legal RT-11 device name.            |
| =   | represents a delimiter, denoting device ownership. |
| B   | represents the background job.                     |
| F   | represents the foreground job.                     |

The dev=F (and dev=B) construction is valid only under the Foreground/Background system. When used under the Single-Job Monitor, the ?ILL DEV? error message occurs.



## System Communication

A device may be owned exclusively by either the foreground or background job. This may be used, for example, to prevent the I/O of two different jobs from being intermixed on the same non-file structured device. For example:

```
.LOAD PP=B,PR,LP=F
```

The papertape punch belongs to the background job while the paper tape reader is available for use by either the background or foreground job; the line printer is owned by the foreground job. All three handlers are made resident in memory.

Different units of the same file-structured device may be owned by different jobs. Thus, for example, DT1 may belong to the background while DT5 may belong to the foreground job. If no ownership is indicated, the device is available for public use.

To change ownership of a device, another LOAD command may be used; it is not necessary to first UNLOAD the device. For example, if RK1 has been assigned to the foreground job as in the example above, the command:

```
.LOA RK1=B
```

reassigns it to the background job.

The system unit of the system device cannot be assigned ownership, and attempts to do so will be ignored. Other units of the same type as the system device, however, can be assigned ownership.

LOAD is valid for use with user-assigned names. For example:

```
.ASSIGN RK2:XY
```

```
.LOA XY=F
```

If the Single-Job, DECTape-based Monitor is being used, loading the necessary device handlers into memory can significantly improve the throughput of the system, since no handlers need to be loaded dynamically (in other words, they need not be loaded, as required, from the DECTape).

|        |
|--------|
| UNLOAD |
|--------|

2.7.2.7 UNLOAD Command - The UNLOAD command is used to make handlers that were previously LOADED non-resident, freeing the memory they were using.



## System Communication

The form of the command is:

UNL{OAD} dev {,dev,...}

where:

dev represents any legal RT-11 device name.

UNLOAD clears ownership for all units of an indicated device type. For example, typing:

.UNL RK2

clears all units of RK. (A request to unload the system device handler clears ownership for any assigned units for that device, but the handler remains resident.)

Any memory freed is returned to a free memory list and eventually reclaimed for the background job after the UNLOAD command is given. UNLOAD is not permitted if the foreground job is running. Such an action might cause a handler which is needed by the foreground job to become non-resident.

Example:

.UNLOAD LP,PP

The lineprinter and paper tape punch handlers are released and the area which they used is freed.

A special function of this command is to remove a terminated foreground job and reclaim memory, since the space occupied by the foreground job is not automatically returned to the free memory list when it finishes. In this instance, the device name FG is used to specify the foreground job. For example:

.UNL FG

FG can be mixed with other device names.

However, if, for example, DT2 has been assigned the name FG and loaded into memory as follows:

.ASSIGN DT2:FG

.LOAD FG

the command:

.UNLOAD FG

causes the foreground job, not DT2, to be unloaded. To unload DT2, this command must be typed:

.UNLOAD DT2



SET

2.7.2.8 SET Command - The SET command is used to change device handler characteristics and certain system configuration parameters.

The form of the command is:

SET dev:{NO}option=value},{NO}option=value,...}

where:

dev: represents any legal RT-11 physical device name.

NO option is the feature or characteristic to be altered.

=value is a decimal number required in some cases.

A space may be used in place of or in addition to the colon, equal sign, or comma. Note that the device indicated (with the exception of TTY) must be a physical device name and is not affected by logical device name assignments which may be active. The name of the characteristic or feature to be altered must be legal for the indicated device (see Table 2-5) and may not be abbreviated.

The SET command locates the file SY:DEV.SYS and permanently modifies it. No modification is done if the command entered is not completely valid. If a handler has already been loaded when a SET command is issued for it, the modifications will not take effect until the handler is unloaded and a fresh copy called in from the system device.

Table 2-5 lists the system characteristics and parameters which may be altered (those modes designated as "normal" are the initial modes):

Table 2-5  
SET Command Options

| Device | Option | Alteration  |
|--------|--------|---|
| LP     | CR     | Allows carriage returns to be sent to the printer. For printers other than the LS11, a line feed performs all the functions of a carriage return, so ignoring carriage returns causes a significant increase in printing speed. The CR option should be used for LS11 line printers, and to obtain the overstriking capability for any line printer. This is the normal mode. |



# System Communication

Table 2-5 (Cont.)  
SET Command Options

| Device | Option  | Alteration   |
|--------|---------|--|
| LP     | NOCR    | Inhibits sending carriage returns to the line printer.   |
| LP     | FORM0   | Causes a form feed to be issued before a request to print block zero. This is the normal mode.   |
| LP     | NOFORM0 | Turns off FORM0 mode.  |
| LP     | HANG    | Causes the handler to wait for user correction if the line printer is not ready or becomes not ready during printing. This is the normal mode.<br><br>New users should note that when expecting output from the line printer and it appears as though the system is not responding or is in an idle state, the line printer should be checked to see if it is on and ready to print. |
| LP     | NOHANG  | Generates an immediate error if the line printer is not ready.   |
| LP     | LC      | Allows lower case characters to be sent to the printer. This option should be used if the printer has a lower case character set.  |
| LP     | NOLC    | Causes lower case characters to be translated to upper case before printing. This is the normal mode.  |
| LP     | WIDTH=n | Sets the line printer width to n, where n is a number between 30 and 255. Any characters printed past column n are ignored. The NO modifier is not permitted.  |
| CR     | CODE=n  | Modifies the card reader handler to use either the DEC 026 or the DEC 029 card codes (refer to Appendix H). n must be either 26 or 29. The NO modifier is not permitted.   |
| CR     | CRLF    | Causes a carriage return/line feed to be appended to each card image. This is the normal mode.   |
| CR     | NOCRLF  | Transfers each card image without appending a carriage return/line feed.   |
| CR     | HANG    | Causes the handler to wait for user correction if the reader is not ready at the start of a transfer. This is the normal mode.   |
| CR     | NOHANG  | Generates an immediate error if the device is not ready at the start of a transfer. Note that the handler will wait regardless of how the option is set if the reader becomes "not ready" during a transfer (i.e., the input hopper is empty, but an end-of-file card has not yet been read).  |



Table 2-5 (Cont.)  
SET Command Options

| Device   | Option  | Alteration   |    |      |      |      |      |      |      |      |      |      |      |      |      |   |   |   |                   |  |  |  |      |      |      |      |      |      |      |      |      |      |      |      |  |  |  |  |    |    |   |   |   |   |   |   |   |   |   |   |
|--|---------|--|----|------|------|------|------|------|------|------|------|------|------|------|------|---|---|---|-------------------|--|--|--|------|------|------|------|------|------|------|------|------|------|------|------|--|--|--|--|----|----|---|---|---|---|---|---|---|---|---|---|
| CR   | IMAGE   | <p>Causes each card column to be stored as a 12-bit binary number, one column per word. The CODE option has no effect in IMAGE mode. The format of the 12-bit binary number is:</p> <p>PDP-11 WORD</p> <table><tr><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td colspan="4">UNUSED (ALWAYS 0)</td><td>ZONE</td><td>ZONE</td><td>ZONE</td><td>ZONE</td><td>ZONE</td><td>ZONE</td><td>ZONE</td><td>ZONE</td><td>ZONE</td><td>ZONE</td><td>ZONE</td><td>ZONE</td></tr><tr><td colspan="4"></td><td>12</td><td>11</td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td></tr></table> <p>This format allows binary card images to be read and is especially useful if a special encoding of punch combinations is to be used. Mark-sense cards may be read in IMAGE mode.</p> | 15 | 14   | 13   | 12   | 11   | 10   | 9    | 8    | 7    | 6    | 5    | 4    | 3    | 2 | 1 | 0 | UNUSED (ALWAYS 0) |  |  |  | ZONE | ZONE | ZONE | ZONE | ZONE | ZONE | ZONE | ZONE | ZONE | ZONE | ZONE | ZONE |  |  |  |  | 12 | 11 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 15   | 14      | 13   | 12 | 11   | 10   | 9    | 8    | 7    | 6    | 5    | 4    | 3    | 2    | 1    | 0    |   |   |   |                   |  |  |  |      |      |      |      |      |      |      |      |      |      |      |      |  |  |  |  |    |    |   |   |   |   |   |   |   |   |   |   |
| UNUSED (ALWAYS 0)  |         |  |    | ZONE | ZONE | ZONE | ZONE | ZONE | ZONE | ZONE | ZONE | ZONE | ZONE | ZONE | ZONE |   |   |   |                   |  |  |  |      |      |      |      |      |      |      |      |      |      |      |      |  |  |  |  |    |    |   |   |   |   |   |   |   |   |   |   |
|  |         |  |    | 12   | 11   | 0    | 1    | 2    | 3    | 4    | 5    | 6    | 7    | 8    | 9    |   |   |   |                   |  |  |  |      |      |      |      |      |      |      |      |      |      |      |      |  |  |  |  |    |    |   |   |   |   |   |   |   |   |   |   |
| CR   | NOIMAGE | <p>Allows the normal translation (as specified by the CODE option) to take place; data is packed one column per byte. Invalid punch combinations are translated into the error character, ASCII "\" (backslash), which is octal code 134. This is the normal mode.</p>   |    |      |      |      |      |      |      |      |      |      |      |      |      |   |   |   |                   |  |  |  |      |      |      |      |      |      |      |      |      |      |      |      |  |  |  |  |    |    |   |   |   |   |   |   |   |   |   |   |
| CR   | TRIM    | <p>Causes trailing blanks to be removed from each card read. It is not recommended that TRIM and NOCRLF be used together since card boundaries will be difficult to find. This is the normal mode.</p>   |    |      |      |      |      |      |      |      |      |      |      |      |      |   |   |   |                   |  |  |  |      |      |      |      |      |      |      |      |      |      |      |      |  |  |  |  |    |    |   |   |   |   |   |   |   |   |   |   |
| CR   | NOTRIM  | <p>Transfers a full 80 characters per card.</p>  |    |      |      |      |      |      |      |      |      |      |      |      |      |   |   |   |                   |  |  |  |      |      |      |      |      |      |      |      |      |      |      |      |  |  |  |  |    |    |   |   |   |   |   |   |   |   |   |   |
| CT   | RAW     | <p>Causes the cassette handler to perform a read-after-write check for every record written, and retry if an output error occurred. If three retries fail, an output error is detected.</p>  |    |      |      |      |      |      |      |      |      |      |      |      |      |   |   |   |                   |  |  |  |      |      |      |      |      |      |      |      |      |      |      |      |  |  |  |  |    |    |   |   |   |   |   |   |   |   |   |   |
| CT   | NORAW   | <p>Causes the cassette handler to write every record directly without reading it back for verification. This significantly increases transfer rates at the risk of increased error rates. Normal mode is NORAW.</p>  |    |      |      |      |      |      |      |      |      |      |      |      |      |   |   |   |                   |  |  |  |      |      |      |      |      |      |      |      |      |      |      |      |  |  |  |  |    |    |   |   |   |   |   |   |   |   |   |   |
| <p>The following options, with the exception of HOLD and NOHOLD, are available in the Foreground/Background System only; HOLD and NOHOLD are available in both systems. These options are not permanent, and must be reissued whenever the monitor is re-bootstrapped. (Note that the device specification is TTY, not TT, because the handler itself is not changed.)</p> |         |  |    |      |      |      |      |      |      |      |      |      |      |      |      |   |   |   |                   |  |  |  |      |      |      |      |      |      |      |      |      |      |      |      |  |  |  |  |    |    |   |   |   |   |   |   |   |   |   |   |
| TTY  | CRLF    | <p>Causes the monitor to issue a carriage return/line feed on the console terminal whenever it attempts to type past the right margin (as set by the WIDTH option). This is the normal mode.</p>   |    |      |      |      |      |      |      |      |      |      |      |      |      |   |   |   |                   |  |  |  |      |      |      |      |      |      |      |      |      |      |      |      |  |  |  |  |    |    |   |   |   |   |   |   |   |   |   |   |
| TTY  | NOCRLF  | <p>Causes no special action to be taken at the right margin.</p>   |    |      |      |      |      |      |      |      |      |      |      |      |      |   |   |   |                   |  |  |  |      |      |      |      |      |      |      |      |      |      |      |      |  |  |  |  |    |    |   |   |   |   |   |   |   |   |   |   |
| TTY  | FB      | <p>Causes the monitor to treat CTRL B and CTRL F characters as background and foreground program control characters and does not transmit them to the user program. This is the normal mode.</p>   |    |      |      |      |      |      |      |      |      |      |      |      |      |   |   |   |                   |  |  |  |      |      |      |      |      |      |      |      |      |      |      |      |  |  |  |  |    |    |   |   |   |   |   |   |   |   |   |   |



# System Communication

Table 2-5 (Cont.)  
SET Command Options

| Device | Option  | Alteration   |
|--------|---------|--|
| TTY    | NOFB    | Causes CTRL B and CTRL F to have no special meaning.<br><br>NOTE<br><br>SET TTY NOFB is issued to KMON, (which runs as a background job) and disables all communication with the foreground job. To enable communication with the foreground job, issue the command SET TTY FB.        |
| TTY    | FORM    | Indicates that the console terminal is capable of executing hardware form feeds.   |
| TTY    | NOFORM  | Causes the monitor to simulate form feeds by typing eight line feeds. This is the normal mode.   |
| TTY    | HOLD    | Enables use of the hold screen mode of operation for the VT50 terminal. The command is a no-op for any terminal other than the VT50. The command is valid for F/B and Single-Job Monitors. Consult the VT50 VIDEO TERMINAL PROGRAMMER'S MANUAL (DEC-00-OVRS-A-D) for more information. |
| TTY    | NOHOLD  | Disables use of the hold screen mode of operation for the VT50 terminal.   |
| TTY    | PAGE    | Causes the monitor to treat CTRL S and CTRL Q characters as terminal output hold and unhold flags, and does not transmit them to the user program. This is the normal mode.  |
| TTY    | NOPAGE  | Causes CTRL S and CTRL Q to have no special meaning.   |
| TTY    | SCOPE   | Causes the monitor to echo RUBOUTs as backspace-space-backspace. This mode should be used when the console is a VT05/VT50 or when GT ON is in effect.  |
| TTY    | NOSCOPE | Causes the monitor to echo RUBOUTs as backslash followed by the character deleted. This is the normal mode.  |
| TTY    | TAB     | Indicates that the console terminal is capable of executing hardware tabs.   |
| TTY    | NOTAB   | Causes the monitor to simulate tab stops every eight positions. The normal mode is NOTAB. VT05/VT50 terminals generally have hardware tabs.  |
| TTY    | WIDTH=n | Sets the width of the console terminal to n positions, for the use of the CRLF option. n must be in the range 30-255 (decimal). The width is initially set to 72.  |



## System Communication

The following variant of the SET command is used to prevent the background job from ever placing the USR in a swapping state (note that USR replaces a device specification in the command line):

SET USR {NO} SWAP

This is useful when running on a DECTape based system, or when running a foreground job which requires the USR but has no memory allocated into which to read it. When the monitor is bootstrapped, it is in the SWAP condition, i.e., the background may place the USR in a swapping state via a SETTOP.

The Single-Job Monitor behaves as though the following options are set: NOTAB, NOFORM, PAGE, NOCRLF, NOSCOPE, NOHOLD.

### 2.7.3 Commands to Manipulate Memory Images

GET

2.7.3.1 GET Command - The GET command (valid for use with a background job only) loads the specified memory image file (not ASCII or binary) into memory from the indicated device.

The form of the GET command is:

GE{T} dev:filnam.ext

where:

|            |   |
|------------|---|
| dev:       | represents any legal RT-11 device name. If a device is not specified, DK: is assumed.                             |
| filnam.ext | represents a valid RT-11 filename and extension. If an extension is not specified, the extension .SAV is assumed. |

The GET command is typically used to load a program into memory for modification and/or debugging. The GET command can also be used in conjunction with the Base, Examine, Deposit, and START commands to test patches, and can be used with SAVE to make patches permanent. Multiple GETs can be used to combine programs. Thus:

|                                 |  |
|---------------------------------|--|
| . GET ODT.SAV                   | Loads ODT into memory                      |
| . GET PROG.                     | Loads PROG.SAV into memory with ODT        |
| . START (ODTs starting address) | Starts execution with ODT (see Chapter 8). |



## System Communication

The GET command cannot be used to load overlay segments of programs; it may only be used to load the root segment (that part which will not be overlaid; refer to Chapter 6, Linker).

Multiple GETs can be used to build a memory image of several programs. If identical locations are required by any of the programs, the later programs overlay the previous ones.

### Examples:

```
. GET DT3:FILE1.SAV  Loads the file FILE1.SAV into memory
                      from DECTape unit 3.

. GET NAME1          Loads the file NAME1.SAV from device DK.
```

## BASE

**2.7.3.2 Base Command** - The B command sets a relocation base. This relocation base is added to the address specified in subsequent Examine or Deposit commands to obtain the address of the location to be referenced. This command is useful when referencing linked modules with the Examine and Deposit commands. The base address can be set to the address where the module of interest is loaded. The form of the command is:

B {location}

where:

location represents an octal address used as a base address for subsequent Examine and Deposit commands.

### NOTE

A space must follow the B command even if an address is not specified (the B<space> command is equivalent to B 0).

Any non-octal digit terminates an address. If location is odd, it is rounded down by one to an even address.

The base is cleared whenever user program execution is initiated.

### Examples:

```
. B          Sets base to 0.
. B 200      Sets base to 200.
. B 201      Sets base to 200.
```



## EXAMINE

2.7.3.3 Examine Command - The E command prints the contents of the specified location(s) in octal on the console terminal. The form of the Examine command is:

E location {,location m-location n}

where:

location represents an octal address which is added to the relocation base value (the value set by the B Command) to get the actual address examined. Any non-octal digit terminates an address. An odd address is truncated to become an even address.

If more than one location is specified (location m-location n), the contents of location m through location n inclusive are printed. The second location specified (location n) must not be less than the first location specified, otherwise an error message is printed. If no location is specified, the contents of location 0 are printed. Examination of locations outside the background area is illegal.

Examples:

. E 1000  
127401

Prints contents of location 1000 (added to the base value if other than 0).

. E 1001-1012  
127401 007624 127400 000000 000000 000000

Prints the contents of locations 1000 (plus the base value if other than 0) through 1013.

## DEPOSIT

2.7.3.4 Deposit Command - The Deposit command deposits the specified value(s) starting at the location given.

The form of the command is:

D location=value1{,value2,...valuen}



## System Communication

where:

**location** represents an octal address which is added to the relocation base value to get the actual address where the values are deposited. Any non-octal digit is accepted as a terminator of an address.

**value** represents the new contents of the location.

If multiple values are specified (value1,...,valuen), they are deposited beginning at the location specified. The DEPOSIT command accepts word or byte addresses but executes the command as though a word address was specified. An odd address is truncated by one to an even address. All values are stored as word quantities.

Any character that is not an octal digit may be used to separate the locations and values in a DEPOSIT command.

An error results when the address specified references a location outside the background job's area.

Examples:

|               |                                  |
|---------------|----------------------------------|
| . D 1000=3705 | Deposits 3705 into location 1000 |
| . B 1000      | Sets relocation base to 1000     |
| . D 1500=2503 | Puts 2503 into location 2500     |
| . B 0         | Resets base to 0                 |

### SAVE

**2.7.3.5 SAVE Command** - The SAVE command writes specified user memory areas to a named file and device in save image format. Memory is written from location 0 to the highest memory address specified by the parameter list.

The SAVE command does not write the overlay segments of programs; it saves only the root segment (refer to Chapter 6, Linker).

The form of the command is:

SAV{E} dev:filnam.ext{parameters}

where:

**dev:** represents one of the standard RT-11 block-replaceable device names. If no device is specified, DK is assumed.



## System Communication

**file.ext** represents the name to be assigned to the file being saved. If the file name is omitted, an error message is output. If no extension is specified, the extension .SAV is used.

**parameters** represent memory locations to be saved. RT-11 transfers memory in 256-word blocks. If the locations specified make a block of less than 256 words, enough additional locations are transferred to make a 256-word block.

Parameters can be specified in the following format:

**areal,area2-arean**

where:

|                    |   |
|--------------------|---|
| <b>areal</b>       | represent an octal number (or numbers separated by dashes). If more than one number is specified, the second number must be greater than the first. |
| <b>area2-arean</b> |   |

The start address and the Job Status Word are given the default value 0 and the stack is set to 1000. If the user wants to change these or any of the following addresses, he must first use the DEPOSIT command to alter them and then SAVE the correct areas:

| <u>Area</u>     | <u>Location</u> |
|-----------------|-----------------|
| Start address   | 40              |
| Stack           | 42              |
| JSW             | 44              |
| USR address     | 46              |
| High address    | 50              |
| Fill characters | 56              |

If the values of the addresses are changed, it is the user's responsibility to reset them to their default values. See Chapter 9 for more information concerning these addresses.

Examples:

```
SAVE FILE1 10000-11000,14000-14100
```

Saves locations 10000(8) through 11777(8) (11000 starts the first word of a new block, therefore the whole block, up to 12000, is stored) and 14000(8) through 14777(8) on DK with the name FILE1.SAV.

```
.SAVE DT1:NAM.NEW 10000
```

Saves locations 10000 through 10777 on DT1: with the name NAM.NEW.

```
.D 44:20000
```

```
.SAV SY:PRAM 1000-5777
```

Sets the reenter bit in the JSW and saves locations 1000 through 5777.



## System Communication

### 2.7.4 Commands to Start a Program

#### RUN

2.7.4.1 RUN Command - The RUN command (valid for use with a background job only) loads the specified memory image file into memory and starts execution at the start address specified in location 40. Under the F/B system, 10 words of user stack area are required to start a user program, and the stack address (location 42) must be initialized to some part of memory where these 10 words will not modify it.

The form of the command is:

$RU\{N\}$  dev:filnam.ext

where:

dev: is any standard device name specifying a block-replaceable device. If dev: is not specified, the device is assumed to be DK. Note that devices MT and CT are not block-replaceable devices and therefore cannot be used in a RUN command.

filnam.ext is the file to be executed. If an extension is not specified, the extension .SAV is assumed.

The RUN command is equivalent to a GET command followed by a START command (with no address specified).

Examples:

|                    |  |
|--------------------|--|
| . RUN DT1:SRCH.SAV | Loads and executes the file SRCH.SAV from DT1.                               |
| . RUN PROG         | Loads PROG.SAV from DK and executes the program.                             |
| . GET PROG1        | Loads PROG1.SAV from device DK without executing it. Then combines PROG1 and |
| . RUN PROG2        | PROG2.SAV in memory and begins execution at the starting address for PROG2.  |



## System Communication

R

**2.7.4.2 R Command** - This command (valid for use with the background job only) is similar to the RUN command except that the file specified must be on the system device (SY:).

The form of the command is:

R filnam.ext

No device may be specified. If an extension is not given, the extension .SAV is assumed.

Examples:

|             |                                     |
|-------------|-------------------------------------|
| . R XYZ.SAV | Loads and executes XYZ.SAV from SY. |
| . R SRC     | Loads and executes SRC.SAV from SY. |

START

**2.7.4.3 START Command** - The START command begins execution of the program currently in memory (i.e., loaded via the GET command) at the specified address. START does not clear or reset memory areas.

The form of the command is:

ST{ART} address

where:

address is an octal number representing any 16-bit address. If the address is omitted, or if 0 is given, the starting address in location 40 will be used.

If the address given does not exist or is not an even address, a trap to location 4 occurs. In this case a monitor error message appears. If no address is given, the program's start address from location 40 is used.



## System Communication

### Examples:

```
.GET FILE.1      Loads FILE.1 into memory and starts execution
                  at location 1000.

.START 1000

.GET FILEA.      Loads FILEA.SAV, then combines FILEA.SAV with
                  FILEB.SAV and starts execution at FILEB's
.GET FILEB.      start address.

.ST
```

## REENTER

2.7.4.4 REENTER Command - The REENTER command starts the program at its reentry address (the start address minus two). REENTER does not clear or reset any memory areas and is generally used to avoid reloading the same program for repetitive execution. It can be used to return to a program whose execution was stopped with a CTRL C.

The form of the command is:

RE{ENTER}

If the reenter bit (bit 13) in the Job Status Word (location 44) is not set, the REENTER command is illegal.

For most system programs, the REENTER command restarts the program at the command level.

If desired, the reentry point in a user program can branch to a routine which initializes the tables and stack, fetches device handlers etc., and then continue normal operation.

### Example:

```
. R PIP          CTRL C interrupts the PIP
*/F             directory listing and transfers
MONITR. SYS     control to the monitor level.
directory prints REENTER returns control to PIP.
.
. (↑C typed)
.
. REENTER
*
```

## 2.7.5 Commands Used Only in a Foreground/Background Environment

It is important to note that in order to control execution of a foreground job, the commands in this section must be typed to KMON, which is running as the background job. Thus, for example, to SUSPEND



## System Communication

the foreground job, the user must be sure he is directing input to KMON as follows:

|            |                                     |
|------------|-------------------------------------|
| F>         | Foreground job is running. Control  |
| (↑B typed) | is redirected to the background job |
| B>         | and PIP is called (the foreground   |
| R PIP      | is still active). CTRL C stops PIP  |
| *^C        | and starts KMON. The foreground     |
| SUSPEND    | job is suspended. (See Section      |
|            | 2.7.5.2.)                           |

FRUN

2.7.5.1 FRUN Command - The FRUN command is used to initiate foreground jobs. FRUN will only run relocatable files produced with the Linker /R switch (using the Linker supplied with RT-11, Version 2). Any handlers used by a foreground job must be in memory.

The form of the command is:

FRU{N} dev:file.ext{/N:n}{/S:n}{/P}

where:

|              |  |
|--------------|--|
| dev:         | represents a block replaceable RT-11 device. If dev: is not specified, DK: is assumed.   |
| file.ext     | represents the job to be executed. The default extension for a foreground job is .REL.   |
| /N:n or /Nin | represents an optional switch used to allocate n words (not bytes) over and above the actual program size. (If running a FORTRAN program, a special formula is used to determine n. Refer to Appendix G for this information.)   |
| /S:n or /Sin | represents an optional switch used to allocate n words (not bytes) for stack space. Normally, stack space is set by default to 128 words and is placed in memory below the program. To change the stack size, use /S:n; the stack is still placed in memory under the program. To relocate the stack area, use an .ASECT (see Chapter 5) to define the start of the user stack in location 42. This overrides the /S switch. |
| /P           | represents an optional switch (at the end of the FRUN command) for debugging purposes. When the carriage return is typed, FRUN prints the load address of the program, but does not start the  |



## System Communication

program. The foreground job must be explicitly started with the RSUME command (see Section 2.7.5.3). For example:

```
.FRUN DATA/P  
LOADED AT 125444
```

If ODT is used with the foreground job, this feature provides the means for determining where the job actually was loaded.

The program is started when the RSUME command is given, allowing the programmer to examine or modify the program before starting it.

If another foreground job is active when the FRUN command is given, an error message is printed. If a terminated foreground job is occupying memory, that region is first reclaimed. Then if the file indicated is found and will fit in memory, the job is installed and started immediately. FRUN destroys the background job's memory image.

### Examples:

|            |  |
|------------|--|
| FRUN F1    | Runs program F1.REL stored on device DK. |
| FRU DT1:F2 | Runs F2.REL which is on DT1.             |

## SUSPEND

**2.7.5.2 SUSPEND Command** - The SUSPEND command is used to stop execution of the foreground job.

The form of the command is:

```
SUS{PEND}
```

No arguments are required. Foreground I/O transfers in progress will be allowed to complete; however, no new I/O requests will be issued and no completion routines will be entered (see Chapter 9 for a discussion of completion routines). Execution of the job can be resumed only from the keyboard.

### Example:

SUSPEND Suspends execution of the foreground job currently running.



## System Communication

### RSUME

**2.7.5.3 RSUME Command** - The RSUME command is used to resume execution of the foreground job where it was suspended. Any completion routines which were scheduled while the foreground was suspended are entered at this time.

The form of the command is:

RSU{ME}

No arguments are required.

Example:

RSU       Resumes execution of the foreground job currently suspended.

## 2.8 MONITOR ERROR MESSAGES

The following error messages are output by the Keyboard Monitor.

| <u>Message</u> | <u>Meaning</u>  |
|----------------|---|
| ?ADDR?         | Address out of range in E or D command.   |
| ?DAT?          | The DATE command argument was illegal, or no argument was given and the date has not yet been set.  |
| ?ER RD OVLY?   | An I/O error occurred while reading a KMON overlay to process the current command. This is a serious error, indicating that the system file MONITR.SYS is unreadable. |
| F?             | A CTRL F was typed and no foreground job exists.  |
| ?F ACTIVE?     | Neither FRUN nor UNLOAD may be used when a foreground job already exists and is active.   |
| ?FIL NOT FND?  | File specified in R, RUN, GET, or FRUN command not found.   |
| ?FILE?         | No file named where one is expected.  |



## System Communication

| <u>Message</u>   | <u>Meaning</u>   |
|------------------|--|
| ?HANDLR?         | Attempted to close a file with no handler in memory. The file cannot be closed.  |
| ?ILL CMD?        | Illegal Keyboard Monitor command or command line too long.   |
| ?ILL DEV?        | Illegal or nonexistent device, or an attempt was made to make a device handler resident for use with a foreground job (dev=F) when the Single-Job Monitor was running. |
| ?NO CLOCK?       | No KWill clock is available for the TIME command.  |
| ?NO FG?          | A SUSPEND, RSUME, or UNLOAD FG command was given, but no foreground job was in memory.   |
| ?OVR COR?        | Attempt to GET or RUN a file that is too big.  |
| ?PARAMS?         | Bad parameters were typed to the SAVE command.   |
| ?REL FIL I/O ER? | Either the program requested is not a REL file or a hardware error was encountered trying to read or write the file.   |
| ?SV FIL I/O ER?  | I/O error on .SAV file in SAVE (output) or R, RUN, GET, or FRUN (input) command.   |
| ?SY I/O ER?      | I/O error on system device (usually reading or writing scratch area).  |
| ?TIM?            | The TIME command argument was illegal.   |

The following messages are output by the RT-11 Resident Monitor when an unrecoverable error has occurred. Control passes to the Keyboard Monitor. The program in which the error occurred cannot be restarted with the RE command. To execute the program again, use the R or RUN command.

The format for fatal monitor error messages is:

?M-text PC                      where PC is the address+2 of the location where the error occurred.

Note that ?M errors can be inhibited in certain cases by the use of the .SERR macro; see Chapter 9 for details.

| <u>Message</u> | <u>Meaning</u>  |
|----------------|---|
| ?M-ILL ADDR    | Under the F/B Monitor, an address specified in a monitor call was odd or was not within the job's limits. |



## System Communication

|                               |  |
|-------------------------------|--|
| ?M-BAD FETCH                  | Either an error occurred while reading in a device handler from SY, or the address at which the handler was to be loaded was illegal.  |
| ?M-DIR IO ERR                 | An error occurred doing I/O in the directory of a device (e.g., .ENTER on a write-locked device).  |
| ?M-DIR OVFL0                  | No more directory segments were available for expansion (occurs during file creation (.ENTER)).  |
| ?M-DIR UNSAFE                 | In F/B only, this message may appear in addition to any of the other diagnostics listed in this section. It indicates that the error occurred while the USR was updating a device directory. One or more files on that device may be lost.                     |
| ?M-FP TRAP                    | A floating-point exception trap occurred, and the user program had no .SFPA exception routine active (see Chapter 9).  |
| ?M-ILL CHAN                   | A channel number was specified which was too large.  |
| ?M-ILL EMT                    | An EMT was executed which did not exist; i.e., the function code was out of bounds.  |
| ?M-ILL USR                    | The USR was called from a completion routine. This error does not have a soft return (i.e., .SERR will not inhibit this message; see Chapter 9).   |
| ?M-NO DEV                     | A READ/WRITE operation was tried but no device handler was in memory for it.   |
| ?M-OVLY ERR                   | A user program with overlays failed to successfully read an overlay.   |
| ?M-SWAP ERR                   | A hard I/O error occurred while the system was attempting to swap KMON or the USR. This is usually caused by a write-locked system device. Under the Single-Job Monitor, this may cause the system to halt.  |
| ?M-TRAP TO 4<br>?M-TRAP TO 10 | The job has referenced illegal memory, an illegal instruction was used, etc. The printed PC indicates where the failure occurred. .SERR will not have an effect on these errors. They can be intercepted using the .TRPSET programmed request (see Chapter 9). |

If CSI errors occur and input was from the console terminal, an error message is printed on the terminal.



## System Communication

| <u>Message</u> | <u>Meaning</u>                   |
|----------------|----------------------------------|
| ?DEV FUL?      | Output file will not fit.        |
| ?FIL NOT FND?  | Input file was not found.        |
| ?ILL CMD?      | Syntax error.                    |
| ?ILL DEV?      | Device specified does not exist. |

### 2.8.1 Monitor HALTS

There are two HALT instructions in the RT-11 V02 monitors, one each in F/B and Single-Job. The Single-Job Monitor will halt only if I/O errors occur during swap operations to the system device. If the S/J Monitor halts, look for a write-locked system device.

The F/B Monitor will halt if a trap to location 4 occurs or if I/O occurs while the system is performing critical operations from which it cannot recover. If the F/B Monitor halts, look for use of non-existent devices, traps from interrupt service routines, or user-corrupted queue elements.

The monitor halts can be detected by their address, which is high in memory, above the resident base address (location 54).



## CHAPTER 3

### TEXT EDITOR

The Text Editor (EDIT) is used to create and modify ASCII source files so that these files can be used as input to other system programs such as the assembler or BASIC. Controlled by user commands from the keyboard, EDIT reads ASCII files from a storage device, makes specified changes and writes ASCII files to a storage device or lists them on the line printer or console terminal. EDIT allows efficient use of VT-11 display hardware, if this is part of the system configuration.

The Editor considers a file to be divided into logical units called pages. A page of text is generally 50-60 lines long (delimited by form feed characters) and corresponds approximately to a physical page of a program listing. The Editor reads one page of text at a time from the input file into its internal buffers where the page becomes available for editing. Editing commands are then used to:

- Locate text to be changed,

- Execute and verify the changes,

- Output a page of text to the output file,

- List an edited page on the line printer or console terminal.

#### 3.1 CALLING AND USING EDIT

To call EDIT from the system device type:

```
R EDIT
```

and the RETURN key in response to the dot (.) printed by the monitor. EDIT responds with an asterisk (\*) indicating it is in command mode and awaiting a user command string.

Type CTRL C to halt the Editor at any time and return control to the monitor. To restart the Editor type .R EDIT or the .REENTER command in response to the monitor's dot. The contents of the buffers are lost when the Editor is restarted.



## Text Editor

### 3.2 MODES OF OPERATION

Under normal usage, the Editor operates in one of two different modes: Command Mode or Text Mode. In Command Mode all input typed on the keyboard is interpreted as commands instructing the Editor to perform some operation. In Text Mode all typed input is interpreted as text to replace, be inserted into, or be appended to the contents of the Text Buffer.

Immediately after being loaded into memory and started, the Editor is in Command Mode. An asterisk is printed at the left margin of the console terminal page indicating that the Editor is waiting for the user to type a command. All commands are terminated by pressing the ALTMODE key twice in succession. Execution of commands proceeds from left to right. Should an error be encountered during execution of a command string, the Editor prints an error message followed by an asterisk at the beginning of a new line indicating that it is still in Command Mode and awaiting a legal command. The command in error (and any succeeding commands) is not executed and must be corrected and retyped.

Text mode is entered whenever the user types a command which must be followed by a text string. These commands insert, replace, exchange, or otherwise manipulate text; after such a command has been typed, all succeeding characters are considered part of the text string until an ALTMODE is typed. The ALTMODE terminates the text string and causes the Editor to reenter Command Mode, at which point all characters are considered commands again.

A special editing mode, called Immediate Mode, can be used whenever the VT-11 display hardware is running. This mode is described in Section 3.7.2.

### 3.3 SPECIAL KEY COMMANDS

The EDIT key commands are listed in Table 3-1. Control commands are typed by holding down the CTRL key while typing the appropriate character.

Table 3-1  
EDIT Key Commands

| Key     | Explanation   |
|---------|---|
| ALTMODE | Echoes \$. A single ALTMODE terminates a text string. A double ALTMODE executes the command string. For example,<br><br>*GMOV A,B\$-1D\$\$  |
| CTRL C  | Echoes at the terminal as ↑C and a carriage return. Terminates execution of EDIT commands, and returns to monitor Command Mode. A double CTRL C is necessary when I/O is in progress. The REENTER command may be used to restart the Editor, but the contents of the text buffers are lost. |

(continued on next page)



Table 3-1 (cont.)  
EDIT Key Commands

| Key    | Explanation  |
|--------|--|
| CTRL O | Echoes ↑O and a carriage return. Inhibits printing on the terminal until completion of the current command string. Typing a second CTRL O resumes output.  |
| CTRL U | Echoes ↑U and a carriage return. Deletes all the characters on the current terminal input line. (Equivalent to typing RUBOUT back to the beginning of the line.)   |
| RUBOUT | Deletes character from the current line; echoes a backslash followed by the character deleted. Each succeeding RUBOUT typed by the user deletes and echoes another character. An enclosing backslash is printed when a key other than RUBOUT is typed. This erasure is done right to left up to the last carriage return/line feed combination. RUBOUT may be used in both Command and Text Modes. |
| TAB    | Spaces to the next tab stop. Tab stops are positioned every eight spaces on the terminal; typing the TAB key causes the carriage to advance to the next tab position.  |
| CTRL X | Echoes ↑X and a carriage return. CTRL X causes the Editor to ignore the entire command string currently being entered. The Editor prints a <CR><LF> and an asterisk to indicate that the user may enter another command. For example:<br><br><pre>*IABCD EFGH^X *</pre> <p>A CTRL U would only cause deletion of EFGH; CTRL X erases the entire command.</p>                                       |

## 3.4 COMMAND STRUCTURE

EDIT commands fall into six general categories:

| <u>Category</u>  | <u>Commands</u> | <u>Section</u> |
|------------------|-----------------|----------------|
| Input/Output     | Edit Backup     | 3.6.1.3        |
|                  | Edit Read       | 3.6.1.1        |
|                  | Edit Write      | 3.6.1.2        |
|                  | End File        | 3.6.1.9        |
|                  | Exit            | 3.6.1.10       |
|                  | List            | 3.6.1.7        |
|                  | Next            | 3.6.1.6        |
|                  | Read            | 3.6.1.4        |
|                  | Verify          | 3.6.1.8        |
|                  | Write           | 3.6.1.5        |
|                  |                 |                |
| Pointer location | Advance         | 3.6.2.3        |
|                  | Beginning       | 3.6.2.1        |
|                  | Jump            | 3.6.2.2        |



## Text Editor

|                   |               |         |
|-------------------|---------------|---------|
| Search            | Find          | 3.6.3.2 |
|                   | Get           | 3.6.3.1 |
|                   | Position      | 3.6.3.3 |
| Text modification | Change        | 3.6.4.4 |
|                   | Delete        | 3.6.4.2 |
|                   | eXchange      | 3.6.4.5 |
|                   | Insert        | 3.6.4.1 |
|                   | Kill          | 3.6.4.3 |
| Utility           | Edit Console  | 3.7.1   |
|                   | Edit Display  | 3.7.1   |
|                   | Edit Version  | 3.6.5.5 |
|                   | Execute Macro | 3.6.5.4 |
|                   | Macro         | 3.6.5.3 |
|                   | Save          | 3.6.5.1 |
| Immediate Mode    | Unsave        | 3.6.5.2 |
|                   | ALTMODE       | 3.7.2   |
|                   | CTRL D        | 3.7.2   |
|                   | CTRL G        | 3.7.2   |
|                   | CTRL N        | 3.7.2   |
|                   | CTRL V        | 3.7.2   |
|                   | RUBOUT        | 3.7.2   |

The general format for the first five categories of EDIT commands is:

nCtext\$  
or  
nC\$

where n represents one of the legal arguments listed in Table 3-2, C is a one- or two-letter command, and text is a string of successive ASCII characters.

As a rule, commands are separated from one another by a single ALTMODE; however, if the command requires no text, the separating ALTMODE is not necessary. Commands are terminated by a single ALTMODE; typing a second ALTMODE begins execution. (ALTMODE is used differently when Immediate Mode is in effect; Section 3.7.2 details its use in this case.)

The format of Display Editor commands is somewhat different from the normal editing command format, and is described in Section 3.7.

### 3.4.1 Arguments

An argument is positioned before a command letter and is used either to specify the particular portion of text to be affected by the command or to indicate the number of times the command should be performed. With some commands, this specification is implicit and no arguments are needed; other editing commands require an argument. Table 3-2 lists the formats of arguments which are used by commands of this second type.



## Text Editor

Table 3-2  
Command Arguments

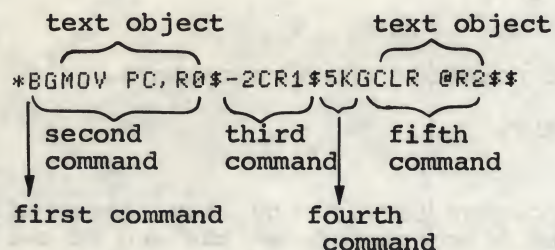
| Format | Meaning  |
|--------|--|
| n      | n stands for any integer in the range -16383 to +16383 and may, except where noted, be preceded by a + or -. If no sign precedes n, it is assumed to be a positive number. Whenever an argument is acceptable in a command, its absence implies an argument of 1 (or -1 if only the - is present). |
| 0      | 0 refers to the beginning of the current line.   |
| /      | / refers to the end of text in the current Text Buffer.  |
| =      | = is used with the J, D and C commands only and represents -n, where n is equal to the length of the last text argument used.  |

The roles of all arguments are explained more specifically in following sections.

### 3.4.2 Command Strings

All EDIT command strings are terminated by two successive ALTMODE characters. Spaces, carriage returns and line feeds within a command string may be used freely to increase command readability but are ignored unless they appear in a text string. Commands used to insert text can contain text strings that are several lines long. Each line is terminated with a <CR><LF> and the entire command is terminated with a double ALTMODE.

Several commands can be strung together and executed in sequence. For example,



Execution of a command string begins when the double ALTMODE is typed and proceeds from left to right. Except when part of a text string, spaces, carriage return, line feed, and single ALTMODES are ignored. For example:

```
*BGM OV R0$=CCLR R1$AV$$
```



## Text Editor

may be typed as:

```
*B$ GMOV R0$  
=CCLR R1$  
A$ V$$
```

with equivalent execution.

### 3.4.3 The Current Location Pointer

Most EDIT commands function with respect to a movable reference pointer which is normally located between the most recent character operated upon and the next character in the buffer. At any given time during the editing procedure, this pointer can be thought of as representing the current position of the Editor in the text. Most commands use this pointer as an implied argument. Commands are available for moving the pointer anywhere in the text, thereby redefining the current location and allowing greater facility in the use of other commands.

### 3.4.4 Character- and Line-Oriented Command Properties

Edit commands are line-oriented or character-oriented depending on the arguments they accept. Line-oriented commands operate on entire lines of text. Character-oriented commands operate on individual characters independent of what or where they are.

When using character-oriented commands, a numeric argument specifies the number of characters that are involved in the operation. Positive arguments represent the number of characters in a forward direction (in relation to the pointer), negative arguments the number of characters in a backward direction. Carriage return and line feed characters are treated the same as any other character. For example, assume the pointer is positioned as indicated in the following text (↑ represents the current position of the pointer):

```
MOV    #VECT,R2<CR><LF>↑  
CLR    @R2<CR><LF>
```

The EDIT command -2J backs the pointer by two characters.

```
MOV    #VECT,R2<CR><LF>  
CLR    @R2<CR><LF>
```

The command 10J advances the pointer forward by ten characters and places it between the CR and LF characters at the end of the second line.

```
MOV    #VECT,R2<CR><LF>  
CLR    @R2<CR><LF>
```

Finally, to place the pointer after the "C" in the first line, a -14J command is used. The J (Jump) command is explained in Section 3.6.2.2.

```
MOV    #VECT,R2<CR><LF>  
CLR    @R2<CR><LF>
```



## Text Editor

When using line-oriented commands, a numeric argument represents the number of lines involved in the operation. The Editor recognizes a line of text as a unit when it detects a <CR><LF> combination in the text. When the user types a carriage return, the Editor automatically inserts a line feed. Positive arguments represent the number of lines forward (in relation to the pointer); this is accomplished by counting carriage return/line feed combinations beginning at the pointer. So, if the pointer is at the beginning of a line, a line-oriented command argument of +1 represents the entire line between the current pointer and the terminating line feed. If the current pointer is in the middle of the line, an argument of +1 represents only the portion of the line between the pointer and the terminating line feed.

For example, assume a buffer of:

```
MOV    PC,R1<CR><LF>
ADD    ↑#DRIV-.,R1<CR><LF>
MOV    #VECT,R2<CR><LF>
CLR    @R2<CR><LF>
```

The command to advance the pointer one line (1A) causes the following change:

```
MOV    PC,R1<CR><LF>
↑ADD    #DRIV-.,R1<CR><LF>
MOV    #VECT,R2<CR><LF>
CLR    @R2<CR><LF>
```

The command 2A moves the pointer over 2 <CR><LF> combinations:

```
MOV    PC,R1<CR><LF>
ADD    #DRIV-.,R1<CR><LF>
MOV    #VECT,R2<CR><LF>
↑CLR    @R2<CR><LF>
```

Negative line arguments reference lines in a backward direction (in relation to the pointer). Consequently, if the pointer is at the beginning of the line, a line argument of -1 means "the previous line" (moving backward past the first <CR><LF> and up to but not including the second <CR><LF>; if the pointer is in the middle of a line, an argument of -1 means the preceding 1 1/2 lines. Assume the buffer contains:

```
MOV    PC,R1<CR><LF>
ADD    #DRIV-.,R1<CR><LF>
MOV    #VECT,R2<CR><LF>
CLR    @R2<CR><LF>
```

A command of -1A backs the pointer by 1 1/2 lines.

```
MOV    PC,R1<CR><LF>
↑ADD    #DRIV-.,R1<CR><LF>
MOV    #VECT,R2<CR><LF>
CLR    @R2<CR><LF>
```



## Text Editor

Now a command of -1A backs it by only 1 line.

```
↑MOV    PC,R1<CR><LF>
  ADD    #DRIV-.,R1<CR><LF>
  MOV    #VECT,R2<CR><LF>
  CLR    @R2<CR><LF>
```

### 3.4.5 Command Repetition

Portions of a command string may be executed more than once by enclosing the desired portion in angle brackets (<>) and preceding the left angle bracket with the number of iterations desired. The structure is:

`C1$C2$n<C3$C4$>C5$$`

where C1, C2,...C5 represent commands and n represents an iteration argument. Commands C1 and C2 are each executed once, then commands C3 and C4 are executed n times. Finally command C5 is executed once and the command line is finished. The iteration argument (n) must be a positive number (1 to 16,383), and if not specified is assumed to be 1. If the number is negative or too large, an error message is printed. Iteration brackets may be nested up to 20 levels. Command lines are checked to make certain the brackets are correctly used and match prior to execution.

Essentially, enclosing a portion of a command string in iteration brackets and preceding it with an iteration argument (n) is equivalent to typing that portion of the string n times. For example:

`*BGAAA$3<-DIB$-J>V$$`

is equivalent to typing:

`*BGAAA$-DIB$-J-DIB$-J-DIB$-JV$$`

and:

`*B3<2<AD>V>$$`

is equivalent to typing:

`*BADADVADADVADADV$$`

The following bracket structures are examples of legal usage:

```
<<><<<><>>>>
<<<>>><><>
```

The following bracket structures are examples of illegal combinations which will cause an error message since the brackets are not properly matched:

```
><><
<<<>>
```

During command repetition, execution proceeds from left to right until a right bracket is encountered. EDIT then returns to the last left



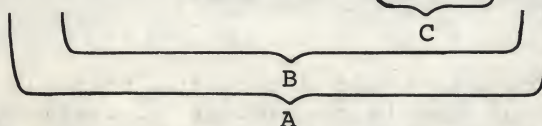
## Text Editor

bracket encountered, decrements the iteration counter and executes the commands within the brackets. When the counter is decremented to 0, EDIT looks for the next iteration count to the left and repeats the same procedures. The overall effect is that EDIT works its way to the innermost brackets and then works its way back again. The most common use for iteration brackets is found in commands such as Unsave, that do not accept repeat counts. For example:

```
*3<U>$$
```

Assume a file called SAMP (stored on device DK) is to be read and the first four occurrences of the instruction MOV #200,R0 on each of the first five pages are to be changed to MOV #244,R4. The following command line is entered:

```
*EBSAMP$5<N4<BGM OV #200,R0$=J$3<G0$=C4$>>>EX$$
```



The command line contains three sets of iteration loops (A,B,C) and is executed as follows:

Execution initially proceeds from left to right; the file SAMP is opened for input, and the first page is read into memory. The pointer is moved to the beginning of the buffer and a search is initiated for the character string MOV #200,R0. When the string is found, the pointer is positioned at the end of the string, but the =J command moves the pointer back so that it is positioned immediately preceding the string. At this point, execution has passed through each of the first two sets of iteration loops (A,B) once. The innermost loop (C) is next executed three times, changing the 0s to 4s. Control now moves back to pick up the second iteration of loop B, and again moves from left to right. When loop C has executed three times, control again moves back to loop B. When loop B has executed a total of 4 times, control moves back to the second iteration of loop A, and so forth until all iterations have been satisfied.

### 3.5 MEMORY USAGE

The memory area used by the Editor is divided into four logical buffers as follows:

|                         |             |
|-------------------------|-------------|
| MACRO BUFFER            | High Memory |
| SAVE BUFFER             |             |
| FREE MEMORY             |             |
| COMMAND INPUT<br>BUFFER | Low Memory  |
| TEXT BUFFER             |             |



## Text Editor

The Text Buffer contains the current page of text being edited, and the Command Input Buffer holds the command currently being typed at the terminal. If a command currently being entered by the user is within 10 characters of exceeding the space available in the Command Buffer, the message:

\* CB ALMOST FULL \*

is printed. If the command can be completed within 10 characters, the user may finish entering the command; otherwise he should type the ALTMODE key twice to execute that portion of the command line already completed. The message is printed each time a character is entered in one of the last 10 spaces.

If the user attempts to enter more than 10 characters the message:

?CB FULL?

is printed and all commands typed within the last 10 characters are ignored. The user again has 10 characters of available space in which to correct the condition.

The Save Buffer contains text stored with the Save (S) command, and the Macro Buffer contains the command string macro entered with the Macro (M) command. (Both commands are explained in Section 3.6.5.)

The Macro and Save Buffers are not allocated space until an M or S command is executed. Once an M or S command is executed, a OM or OU (Unsave) command must be executed to return that space to the free area.

The size of each buffer automatically expands and contracts to accommodate the text being entered; if there is not enough space available to accommodate required expansion of any of the buffers, a "?\*NO ROOM\*?" error message is typed.

## 3.6 EDITING COMMANDS

### 3.6.1 Input/Output Commands

Input commands are used to create files and read them into the Text Buffer where they become available for editing or listing. Output commands cause text to be listed on the console terminal or line-printer or written out to a storage device. Some commands are specifically designed for either input or output functions, while a few commands serve both purposes.

Once editing is completed and the page currently in the Text Buffer is written to the output file, that page of text is unavailable for further editing until the file is closed and reopened.

3.6.1.1 Edit Read - The Edit Read command opens an existing file for input and prepares it for editing.



## Text Editor

The form of the command is:

ERdev:filnam.ext\$

The string argument (dev:filnam.ext) is limited to 19 characters and specifies the file to be opened. If no device is specified, DK: is assumed. If a file is currently open for input, that file is closed; any edits made to the file are preserved.

Edit Read does not input a page of text nor does it affect the contents of the other user buffers (see Section 3.5.)

Edit Read can be used on a file which is already open to close that file for input and reposition EDIT at its beginning. The first Read command following any Edit Read command inputs the first page of the file.

Examples:

\*ERDT1:SAMP.MAC\$\$ Opens SAMP.MAC on device DT1: for input.

\*ERSOURCE\$\$ Opens SOURCE on device DK: for input.

3.6.1.2 Edit Write - The Edit Write command sets up a file for output of newly created or edited text. However, no text is output and the contents of the user buffers are not affected. Any current output files are closed.

The form of the command is:

EWdev:filnam.ext[n]\$

The string argument (dev:filnam.ext[n]) is limited to 19 characters and is the name to be assigned to the output file being opened. If dev: is not specified, DK: is assumed. [n] is optional and represents the length of the file to be opened. If not specified, the largest available space is used.

If a file with the same name already exists on the device, the old file is deleted when an EXit, End File or another Edit Write command is executed.

Examples:

\*EWDK:TEST.MAC\$\$ Opens the file TEST.MAC on device DK: for output.

\*EWFILE.BAS[11]\$\$ Opens the file FILE.BAS (allocating 11 blocks) on the device DK: for output.

3.6.1.3 Edit Backup - The Edit Backup command is used to open an existing file for editing and at the same time create a backup version of the file. No text is read or written with this command.



## Text Editor

The form of the command is:

EBdev:filnam.ext[n]\$

The device designation, filename and extension are limited to 19 characters. If dev: is not specified, DK: is assumed. [n] is optional and represents the length of the file to be opened; if not specified, one-half the largest available space is used.

The file indicated in the command line must already exist on the device designated since text will be read from this file as input. At the same time, an output file is opened under the same filename and extension. After an EB command has been successfully executed, the original file (used as input) is renamed with the current filename and a .BAK extension; any previous file with this filename and a .BAK extension is deleted. The new output file is closed and assigned the name as specified in the EB command. This renaming of files takes place whenever an Exit, End File, Edit Read, Edit Write or Edit Backup command is executed.

### Examples:

|                     |   |
|---------------------|---|
| *EBSY:BAS1.MAC\$\$  | Opens BAS1.MAC on SY. When editing is complete, the old BAS1.MAC becomes BAS1.BAK and the new file becomes BAS1.MAC. Any previous version of BAS1.BAK is deleted.                           |
| *EBBAS2.BAS[15]\$\$ | Opens BAS2.BAS on DK (allocating 15 blocks). When editing is complete, the old BAS2.BAS is labeled BAS2.BAK and the new file becomes BAS2.BAS. Any previous version of BAS2.BAK is deleted. |

In EB, ER and EW commands, leading spaces between the command and the filename are illegal (the filename is considered to be a text string). All dev:file.ext specifications for EB, ER and EW commands conform to the RT-11 conventions for file naming and are identical to filenames entered in command strings used with other system programs.

3.6.1.4 Read - The Read command (R) causes a page of text to be read from the input file (previously specified in an ER or EB command) and appended to the current contents, if any, of the Text Buffer.

The form of the command is:

R

No arguments are used with the R command and the pointer is not moved. Text is input until one of the following conditions is met:

1. A form feed character, signifying the end of the page, is encountered. At this point, the form feed will be the last character in the buffer; or



## Text Editor

2. The Text Buffer is within 500 characters of being full. (When this condition occurs, Read inputs up to the next <CR><LF> combination, then returns to Command Mode. An asterisk is printed as though the Read were complete, but text will not have been fully input); or
3. An end-of-file condition is detected, (the \*EOF\* message is printed when all text in the file has been read into memory and no more input is available).

The maximum number of characters which can be brought into memory with an R command is approximately 6,000 for an 8K system. Each additional 4K of memory allows approximately 8,000 additional characters to be input. An error message is printed if the Read exceeds the memory available or if no input is available.

3.6.1.5 Write - The Write command (W) moves lines of text from the Text Buffer to the output file (as specified in the EW or EB command). The format of the command is:

- nW Write all characters beginning at the pointer and ending at the nth <CR><LF> to the output file.
- nW Write all characters beginning on the -nth line and terminating at the pointer to the output file.
- OW Write the text from the beginning of the current line to the pointer.
- /W Write the text from the pointer to the end of the buffer.

The pointer is not moved and the contents of the buffer are not affected. If the buffer is empty when the Write is executed, no characters are output.

### Examples:

- \*5W\$\$ Writes the next 5 lines of text starting at the pointer, to the current output file.
- \*-2W\$\$ Writes the previous 2 lines of text, ending at the pointer, to the current output file.
- \*B/W\$\$ Writes the entire Text Buffer to the current output file.



## Text Editor

3.6.1.6 Next - The Next command acts as both an input and output command since it performs both functions. First it writes the current Text Buffer to the output file, then clears the buffer, and finally reads in the next page of the input file. The Next command can be repeated n times by specifying an argument before the command. The command format is:

nN

Next accepts only positive arguments (n) and leaves the pointer at the beginning of the buffer. If fewer than n pages are available in the input file, all available pages are input to the buffer, output to the current file, and deleted from the buffer; the pointer is left positioned at the beginning of an empty buffer, and an error message is printed. This command is equivalent to a combination of the Beginning, Write, Delete and Read commands (B/W/DR). Next can be used to space forward, in page increments, through the input file.

Example:

|         |  |
|---------|--|
| *2N\$\$ | Writes the contents of the current Text Buffer to the output file. Read and write the next page of text. Clear the buffer and then read in another page. |
|---------|--|

3.6.1.7 List - The List command prints the specified number of lines on the console terminal. The format of the command is:

|     |  |
|-----|--|
| nL  | Print all characters beginning at the pointer and ending with the nth <CR><LF>.                          |
| -nL | Print all characters beginning with the first character on the -nth line and terminating at the pointer. |
| 0L  | Print from the beginning of the current line up to the pointer.  |
| /L  | Print from the pointer to the end of the buffer.   |

The pointer is not moved after the command is executed.

Examples:

|          |  |
|----------|--|
| *-2L\$\$ | Prints all characters starting at the second preceding line and ending at the pointer. |
| *4L\$\$  | Prints all characters beginning at the pointer and terminating at the 4th <CR><LF>.    |

Assuming the pointer location is:

```
MOVB 5(R1),@R2
ADD↑ R1,(R2)+
```



## Text Editor

The command:

```
*-1L$$
```

Prints the previous 1 1/2 lines up to the pointer:

```
MOVB 5(R1),@R2  
ADD
```

3.6.1.8 Verify - The Verify command prints the current text line (the line containing the pointer) on the terminal. The position of the pointer within the line has no effect and the pointer does not move. The command format is:

V

No arguments are used. The V command is equivalent to a OLL (List) command.

Example:

```
*V$$  
ADD R1,(R2)+
```

The command causes the current line of text to be printed.

3.6.1.9 End File - The End File command closes the current output file. This command does no input/output operations and does not move the pointer. The buffer contents are not affected. The output file is closed, containing only the text previously output.

The form of the command is:

EF

No arguments are used. Note that an implied EF command is included in EW and EB commands.

3.6.1.10 EXit - The EXit command is used to terminate editing, copy the remainder of the input file to the output file, and return control to the monitor. It performs consecutive Next commands until the end of the input file is reached, then closes both the input and output files.

The command format is:

EX

No arguments are used. Essentially, Exit is used to copy the remainder of the input file into the output file and return to the monitor. Exit is legal only when there is an output file open. If an output file is not open and it is desired to terminate the editing session, return to the monitor with CTRL C.



## Text Editor

### NOTE

An EF or EX command is necessary in order to make an output file permanent. If CTRL C is used to return to the monitor without a prior execution of an EF command, the current output file is not saved. (It can however, be made permanent using the monitor CLOSE command; see Section 2.7.2.5.)

An example of the contrasting uses of the EF and EX commands follows. Assume an input file, SAMPLE, contains several pages of text. The user wishes to make the first and second pages of the file into separate files called SAM1 and SAM2, respectively; the remaining pages of text will then make up the file SAMPLE. This can be done using these commands:

```
*EWSAM1$$  
*ERSAMPLE$$  
*RNEF$$  
*EWSAM2$$  
*NEF$$  
*EWSAMPLE$EX$$
```

The user might note that the EF commands are not necessary in this example since the EW command closes a currently open output file before opening another.

### 3.6.2 Pointer Relocation Commands

Pointer relocation commands allow the current location pointer to be moved within the Text Buffer.

3.6.2.1 Beginning - The Beginning command moves the current location pointer to the beginning of the Text Buffer.

The command format is:

B

There are no arguments.

For example, assume the buffer contains:

```
MOVB 5(R1),@R2  
ADD R1,(R2)+  
CLR @R2  
MOVB 6(R1),@R2
```



## Text Editor

The B command:

\*B\$\$

moves the pointer to the beginning of the Text Buffer:

```
↑ MOVB 5(R1),@R2
  ADD  R1,(R2)+
  CLR  @R2
  MOVB 6(R1),@R2
```

3.6.2.2 Jump - The Jump command moves the pointer over the specified number of characters in the Text Buffer.

The form of the command is:

|             |   |
|-------------|---|
| (+ or -) nJ | Move the pointer (backward or forward) n characters.  |
| 0J          | Move the pointer to the beginning of the current line (equivalent to 0A).                         |
| /J          | Move the pointer to the end of the Text Buffer (equivalent to /A).                                |
| =J          | Move the pointer backward n characters, where n equals the length of the last text argument used. |

Negative arguments move the pointer toward the beginning of the buffer, positive arguments toward the end. Jump treats carriage return, line feed, and form feed characters the same as any other character, counting one buffer position for each.

Examples:

|                  |   |
|------------------|---|
| *3J\$\$          | Moves the pointer ahead three characters.   |
| *-4J\$\$         | Moves the pointer back four characters.   |
| *B\$GABC\$=J\$\$ | Move the pointer so that it immediately precedes the first occurrence of 'ABC' in the buffer. |

3.6.2.3 Advance - The Advance command is similar to the Jump command except that it moves the pointer a specified number of lines (rather than single characters) and leaves it positioned at the beginning of the line.

The form of the command is:

|    |   |
|----|---|
| nA | Advance the pointer forward n lines and position it at the beginning of the nth line. |
|----|---|



## Text Editor

|     |   |
|-----|---|
| -nA | Move the pointer backward past n<br><CR><LF> combinations and position it at<br>the beginning of the -nth line. |
| 0A  | Advance the pointer to the beginning of<br>the current line (equivalent to 0J).                                 |
| /A  | Advance the pointer to the end of the<br>Text Buffer (equivalent to /J).  |

### Examples:

\*3A\$\$ Moves the pointer ahead three lines.

Assuming the buffer contains:

CLR @R2  
↑

The command:

\*0A\$\$

Moves the pointer to:

↑CLR @R2

### 3.6.3 Search Commands

Search commands are used to locate specific characters or strings of characters within the Text Buffer.

**3.6.3.1 Get -** The Get command starts at the pointer and searches the current Text Buffer for the nth occurrence of a specified text string. If the search is successful, the pointer is left immediately following the nth occurrence of the text string. If the search fails, an error message is printed and the pointer is left at the end of the Text Buffer. The format of the command is:

nGtext\$

The argument (n) must be positive and is assumed to be 1 if not otherwise specified. The text string may be any length and immediately follows the G command. The search is made on the portion of the text between the pointer and the end of the buffer.

Example:

Assuming the buffer contains:

```
↑MOV    PC,R1
  ADD    #DRIV-.,R1
  MOV    #VECT,R2
  CLR    @R2
  MOVB   5(R1),@R2
  ADD    R1,(R2)+
  CLR    @R2
  MOVB   6(R1),@R2
```



## Text Editor

The command:

```
*GADD$$
```

positions the pointer at:

```
ADD #DRIV-. ,R1
```

The command:

```
*3G@R2$$
```

positions the pointer at:

```
ADD R1,(R2)+  
CLR @R2+
```

After search commands, the pointer is left immediately following the text object. Using a search command in combination with =J will place the pointer before the text object, as follows:

```
*GTEST$=J$$
```

This command combination places the pointer before 'TEST'.

3.6.3.2 Find - The Find command starts at the current pointer and searches the entire input file for the nth occurrence of the text string. If the nth occurrence of the text string is not found in the current buffer, a Next command is automatically performed and the search is continued on the new text in the buffer. When the search is successful, the pointer is left immediately following the nth occurrence of the text string. If the search fails (i.e., the end-of-file is detected for the input file and the nth occurrence of the text string has not been found), an error message is printed and the pointer is left at the beginning of an empty Text Buffer.

The form of the command is:

```
nFtext$
```

The argument (n) must be positive and is assumed to be 1 if not otherwise specified.

By deliberately specifying a nonexistent search string, the user can close out his file; that is, he can copy all remaining text from the input file to the output file.

Find is a combination of the Get and Next commands.

Example:

```
*2FMOVB 6(R1),@R2$$
```

Searches the entire input file for the second occurrence of the text string MOVB 6(R1),@R2. Each unsuccessfully searched buffer is written to the output file.



## Text Editor

3.6.3.3 Position - The Position command searches the input file for the nth occurrence of the text string. If the desired text string is not found in the current buffer, the buffer is cleared and a new page is read from the input file. The format of the command is:

nPtext\$

The argument (n) must be positive, and is assumed to be 1 if not otherwise specified. When a P command is executed the current contents of the buffer are searched from the location of the pointer to the end of the buffer. If the search is unsuccessful, the buffer is cleared and a new page of text is read and the cycle is continued.

If the search is successful, the pointer is positioned after the nth occurrence of the text. If it is not, the pointer is left at the beginning of an empty Text Buffer.

The Position command is a combination of the Get, Delete and Read commands; it is most useful as a means of placing the location pointer in the input file. For example, if the aim of the editing session is to create a new file from the second half of the input file, a Position search will save time.

The difference between the Find and Position commands is that Find writes the contents of the searched buffer to the output file while Position deletes the contents of the buffer after it is searched.

Example:

```
*PADD R1.(R2)+$$    Searches the entire input file for the
                     specified string ignoring the
                     unsuccessfully searched buffers.
```

## 3.6.4 Text Modification Commands

The following commands are used to insert, relocate, and delete text in the Text Buffer.

3.6.4.1 Insert - The Insert command causes the Editor to enter Text Mode and allows text to be inserted immediately following the pointer. Text is inserted until an ALTMODE is typed and the pointer is positioned immediately after the last character of the insert. The command format is:

Itext\$

No arguments are used with the Insert command, and the text string is limited only by the size of the Text Buffer and the space available. All characters except ALTMODE are legal in the text string. ALTMODE terminates the text string.

### NOTE

Forgetting to type the I command will cause the text entered to be executed as commands.



## Text Editor

EDIT automatically protects against overflowing the Text Buffer during an Insert. If the I command is the first command in a multiple command line, EDIT ensures that there will be enough space for the Insert to be executed at least once. If repetition of the command exceeds the available memory, an error message is printed.

### Example:

|       |                |                                |
|-------|----------------|--------------------------------|
| *IMOV | #BUFF, R2      | Inserts the specified text at  |
| MOV   | #LINE, R1      | the current location of the    |
| MOVB  | -1(R2), R0\$\$ | pointer and leaves the pointer |
| *     |                | positioned after R0.           |

3.6.4.2 Delete - The Delete command removes a specified number of characters from the Text Buffer. Characters are deleted starting at the pointer; upon completion, the pointer is positioned at the first character following the deleted text.

The form of the command is:

|             |   |
|-------------|---|
| (+ or -) nD | Delete n characters (forward or backward from the pointer).                     |
| 0D          | Delete from beginning of current line to the pointer (equivalent to 0K).        |
| /D          | Delete from pointer to end of Text Buffer (equivalent to /K).                   |
| =D          | Delete -n characters, where n equals the length of the last text argument used. |

### Examples:

|                   |   |
|-------------------|---|
| *-2D\$\$          | Deletes the two characters immediately preceding the pointer.   |
| *B\$FM0V R1\$=D\$ | Deletes the text string 'MOV R1'. (=D used in combination with a search command will delete the indicated text string). |

Assuming a buffer of:

```
ADD      R1, (R2)+
CLR      ↑@R2
```

the command:

```
*0D$$
```

leaves the buffer with:

```
ADD      R1, (R2)+
↑@R2
```



## Text Editor

3.6.4.3 Kill - The Kill command removes n lines from the Text Buffer. Lines are deleted starting at the location pointer; upon completion of the command, the pointer is positioned at the beginning of the line following the deleted text. The command format is:

|     |   |
|-----|---|
| nK  | Delete lines beginning at the pointer and ending at the nth <CR><LF>.                       |
| -nK | Delete lines beginning with the first character in the -nth line and ending at the pointer. |
| OK  | Delete from the beginning of the current line to the pointer (equivalent to 0D).            |
| /K  | Delete from the pointer to the end of the Text Buffer (equivalent to /D).                   |

Example:

|         |   |
|---------|---|
| *2K\$\$ | Delete lines starting at the current location pointer and ending at the 2nd <CR><LF>. |
|---------|---|

Assuming a buffer of:

|                  |             |
|------------------|-------------|
| ADD              | R1, (R2) +  |
| CLR <sub>1</sub> | @R2         |
| MOVB             | 6 (R1), @R2 |

the command:

\* /K \$\$

alters the contents of the buffer to:

|                  |            |
|------------------|------------|
| ADD              | R1, (R2) + |
| CLR <sub>1</sub> |            |

Kill and Delete commands perform the same function, except that Kill is line-oriented and Delete is character-oriented.

3.6.4.4 Change - The Change command replaces n characters, starting at the pointer, with the specified text string and leaves the pointer positioned immediately following the changed text.

The form of the command is:

|                   |   |
|-------------------|---|
| (+ or -) nCtext\$ | Replace n characters (forward or backward from the pointer) with the specified text.                                |
| 0Ctext\$          | Replace the characters from the beginning of the line up to the pointer with the specified text (equivalent to 0X). |
| /Ctext\$          | Replace the characters from the pointer to the end of the buffer with the specified text (equivalent to /X).        |



## Text Editor

=Ctext\$    Replace -n characters with the indicated text string, where n represents the length of the last text argument used.

The size of the text is limited only by the size of the Text Buffer and the space available. All characters are legal except ALTMODE which terminates the text string.

If the C command is to be executed more than once (i.e., it is enclosed in angle brackets) and if there is enough space available so that the command can be entered, it will be executed at least once (provided it appears first in the command string). If repetition of the command exceeds the available memory, an error message is printed. The Change command is identical to executing a Delete command followed by an Insert (nDitext\$).

Examples:

\*5C#VECT\$\$                      Replaces the five characters to the right of the pointer with #VECT.

Assuming a buffer of:

```
CLR      @R2
MOV↑     5(R1),@R2
```

The command:

\*0CADDB\$\$

leaves the buffer with:

```
CLR      @R2
ADDB↑    5(R1),@R2
```

=C can be used in conjunction with a search command to replace a specific text string as follows:

\*GFIFTY:=\$=CFIVE:\$    Find the occurrence of the text string FIFTY: and replace it with the text string FIVE:.

3.6.4.5 Exchange - The Exchange command exchanges n lines, beginning at the pointer, with the indicated text string and leaves the pointer positioned after the changed text.

The form of the command is:

nXtext\$    Exchange all characters beginning at the pointer and ending at the nth <CR><LF> with the indicated text.

-nXtext\$    Exchange all characters beginning with the first character on the -nth line and ending at the pointer with the indicated text.

0Xtext\$    Exchange the current line from the beginning to the pointer with the specified text (equivalent to 0C).



## Text Editor

/Xtext\$ Exchange the lines from the pointer to the end of the buffer with the specified text (equivalent to /C).

All characters are legal in the text string except ALTMODE which terminates the text.

The Exchange command is identical to a Kill command followed by an Insert (nKIttext\$), and accepts all legal line-oriented arguments.

If the X command is enclosed within angle brackets so that it will be executed more than once, and if there is enough memory space available so that the X command can be entered, it will be executed at least once (provided it is first in the command string). If repetition of the command exceeds the available memory, an error message is printed.

Example:

|                 |                                   |
|-----------------|-----------------------------------|
| *2XADD R1,(R2)+ | Exchanges the two lines to        |
| CLR @R2         | the right of the pointer location |
| \$\$            | with the text string.             |
| *               |                                   |

### 3.6.5 Utility Commands

3.6.5.1 Save - The Save command starts at the pointer and copies the specified number of lines into the Save Buffer (described previously in Section 3.5).

The form of the command is:

nS

The argument (n) must be positive. The pointer position does not change and the contents of the Text Buffer are not altered. Each time a Save is executed, the previous contents of the Save Buffer, if any, are destroyed. If the Save command causes an overflow of the Save Buffer, an error message is printed.

Example:

Assume the Text Buffer contains the following assembly language subroutine:



## Text Editor

```
;SUBROUTINE MSGTYP
;WHEN CALLED, EXPECTS R0 TO POINT TO AN
;ASCII MESSAGE THAT ENDS IN A ZERO BYTE,
;TYPES THAT MESSAGE ON THE USER TERMINAL

MSGTYP:      .ASECT
             .=1000
             TSTB (%0)           ;DONE?
             BEQ MDONE           ;YES-RETURN
MLOOP:      TSTB @#177564        ;NO-IS TERMINAL READY?
             BPL MLOOP          ;NO-WAIT
             MOVB (%0)+,@#177566 ;YES PRINT CHARACTER
             BR MSGTYP           ;LOOP
MDONE:      RTS %7              ;RETURN
```

The command:

```
*145$$
```

stores the entire subroutine in the Save Buffer; it may then be inserted in a program wherever needed by using the U command.

3.6.5.2 Unsave - The Unsave command inserts the entire contents of the Save Buffer into the Text Buffer at the pointer location and leaves the pointer positioned following the inserted text.

The form of the command is:

U     Insert in the Text Buffer the contents of the Save Buffer.

OU    Clear the Save Buffer and reclaim the area for text.

Zero is the only legal argument to the U command.

The contents of the Save Buffer are not destroyed by the Unsave command (only by the OU command) and may be Unsaved as many times as desired.

If there is no text in the Save Buffer and the U command is given, the ?\*NO TEXT\*? error message is printed. If the Unsave command causes an overflow of the Text Buffer, the ?\*NO ROOM\*? error message is displayed.

3.6.5.3 Macro - The Macro command inserts a command string into the EDIT Macro Buffer. The Macro command is of the form:

|                   |   |
|-------------------|---|
| M/command string/ | Store the command string in the Macro Buffer.         |
| OM                | Clear the Macro Buffer and reclaim the area for text. |
| or                |   |
| M//               |   |

/ represents the delimiter character. The delimiter is always the first character following the M command, and may be any character which does not appear in the Macro command string itself.



## Text Editor

Starting with the character following the delimiter, EDIT places the Macro command string characters into its internal Macro Buffer until the delimiter is encountered again. At this point, EDIT returns to Command Mode. The Macro command does not execute the Macro string; it merely stores the command string so that it can be executed later by the Execute Macro (EM) command. Macro does not affect the contents of the Text or Save Buffers.

All characters except the delimiter are legal Macro command string characters, including single ALTMODES to terminate text commands. All commands, except the M and EM commands, are legal in a command string macro.

In addition to the OM command, typing the M command immediately followed by two identical characters (assumed to be delimiters) and two ALTMODE characters also clears the Macro Buffer.

### Examples:

|                                 |                                    |
|---------------------------------|------------------------------------|
| <code>*M//\$\$</code>           | Clears the Macro Buffer            |
| <code>*M/GR0\$-C1\$/\$\$</code> | Stores a Macro to change R0 to R1. |

### NOTE

Be careful to choose infrequently used characters as macro delimiters; use of frequently used characters can lead to inadvertent errors. For example,

```
*M GMOV R0$=CADD R1$ $$  
?*NO FILE*?
```

In this case, it was intended that the macro be `GMOV R0$=CADD R1$` but since the delimiter character (the character following the M) is a space, the space following MOV is used as the second delimiter, terminating the macro. EDIT then returns an error when the `R0$=` becomes an illegal command structure.

**3.6.5.4 Execute Macro -** The Execute Macro command executes the command string specified in the last Macro command.

The form of the command is:

`nEM`

The argument (n) must be positive. The macro is executed n times and returns control to the next command in the original command string.



## Text Editor

### Examples:

```
*M/BGR0$-C1$/$$  
*B1000EM$$  
?*SRCH FAIL IN MACRO*?  
*
```

Executes the MACRO stored in the previous example. An error message is returned when the end of buffer is reached. (This macro effectively changes all occurrences of R0 in the Text Buffer to R1.)

```
*IMOV PC,R1$2EMICLR @R2$$  
*
```

In a new program, inserts MOV PC,R1 then executes the command in the Macro Buffer twice before inserting CLR @R2.

3.6.5.5 Edit Version - The Edit Version command displays the version number of the Editor in use on the console terminal.

The form of the command is:

EV\$

### Example:

```
*EV$$  
V02-01  
*
```

## 3.7 THE DISPLAY EDITOR

In addition to all functions and commands mentioned thus far, the Editor has additional capabilities to allow efficient use of VT-11 display hardware which may be part of the system configuration (GT40, GT44, DECLAB 11/40).

The most apparent feature is the ability to use the display screen rather than the console terminal as a window into the Text Buffer for printout of all textual input and output. When all the features of the display Editor are in use, the screen displays text as shown in Figure 3-1:



## Text Editor



Figure 3-1  
Display Editor Format

The major advantage is that the user can now see immediately where the pointer is. The pointer appears between characters on the screen as a bright blinking L-shaped cursor and can be detected easily and quickly. Note that if the pointer is placed between a carriage return and line feed, it appears in an inverted position at the beginning of the next line.

In addition to displaying the current line (the line containing the cursor), the 10 lines of text preceding the current line and the 9 lines following it are also in view. Each time a command string is executed (via a double ALTMODE) this portion of the screen is refreshed so that it reflects the results of the commands just performed.

The lower section of the screen contains 4 lines of editing commands. The command line currently being entered is last, preceded by the three most recent command lines. This section is separated from the text portion of the screen by a horizontal line of dashes. As new command lines are entered, previous command lines are scrolled upward off the command section so that only four command lines are ever in view.

### 3.7.1 Using the Display Editor

The display features of the Editor are automatically invoked whenever the system scroller is in use and the user types:

```
R EDIT
```

However, if the system does not contain VT-11 display hardware, the display features are not enabled.



## Text Editor

Providing that the system does contain VT-11 display hardware and that the user wishes to employ the screen during the editing session, he may activate it in one of two ways (all editing commands and functions previously discussed in this chapter are valid for use):

1. If the scroller is in use (i.e., the GT ON monitor command has been typed prior to calling the Editor), EDIT recognizes this and automatically continues using the screen for display of text and commands. However, it rearranges the scroller so that a "window" into the Text Buffer appears in the top two-thirds of the screen, while the bottom third is used to display command lines. This arrangement is shown in Figure 3-1.

The Edit Console command can be used to return the scroller to its normal mode so that text and commands appear as described in Chapter 2, Section 2.7.1 (i.e., using the full screen for display of command lines, and eliminating the window). The form of the command is:

EC

For example:

\*BAEC2L\$\$

The second and third lines of the current buffer are listed on the screen; there is no window into the Text Buffer at this point.

Subsequent EC commands are ignored if the window into the Text Buffer is not being displayed.

To recall the window, the Edit Display command is used:

ED

The screen is again arranged as shown in Figure 3-1.

2. Assume the scroller is not in use (i.e., the GT ON command has not been typed, or the monitor GT OFF command has been typed prior to calling the Editor). When the user calls EDIT, an asterisk appears on the console terminal as described in Section 3.1. Using the ED command at this time provides the window into the Text Buffer; however, commands continue to be echoed to the console terminal.

When ED is used in this case, it must be the first command issued; otherwise, it becomes an illegal command (since the memory used by the display buffer and code, amounting to over 600 words, is reclaimed as working space). The display cannot be used again until a fresh copy of EDIT is loaded.

While the display of the text window is active, ED commands are ignored.

Typing the EC command clears the screen and returns all output to the console terminal.



## Text Editor

### NOTE

Under the Single-Job Monitor only, after the editing session is over, it is recommended that the screen be cleared by either typing the EC command, or returning to the monitor and using the monitor INITIALIZE command. Failure to do this may cause unpredictable results.

### 3.7.2 Setting the Editor to Immediate Mode

An additional mode is available in EDIT to provide an easier and faster degree of interaction during the editing session. This mode is called Immediate Mode and combines the most-used functions of the Text and Command Modes--namely, to reposition the pointer and to delete and insert characters.

Immediate Mode may be used only when the VT-11 display hardware is active and the Editor is running; it is entered by typing two ALTMODES (only) in response to the Command Mode asterisk:

\*\$\$

The Editor responds by echoing an exclamation point on the screen. The exclamation character remains on the screen as long as control remains in Immediate Mode.

Once Immediate Mode has been entered, only the commands in Table 3-3 are used. None of these commands echoes, but the text appearing on the screen is constantly refreshed and updated during the editing process. Note that no EDIT commands other than those in Table 3-3 may be used while control remains in Immediate Mode.

To return control to the display Editor's normal Command Mode at any time while in Immediate Mode, type a single ALTMODE. The Editor responds with an asterisk and the user may proceed using all normal Editing commands. (Immediate Mode commands typed at this time will be accepted as Command Mode input characters.) To return control to the monitor while in Immediate Mode, type CTRL C.

Table 3-3  
Immediate Mode Commands

| Command | Meaning   |
|---------|---|
| CTRL N  | Advance the pointer (cursor) to the beginning of the next line (equivalent to A).   |
| CTRL G  | Move the pointer (cursor) to the beginning of the previous line (equivalent to -A). |

(continued on next page)



## Text Editor

Table 3-3 (Cont.)  
Immediate Mode Commands

| Command                                 | Meaning  |
|---|--|
| CTRL D                                  | Move the pointer (cursor) forward by one character (equivalent to J).                              |
| CTRL V                                  | Move the pointer (cursor) back by one character (equivalent to -J).                                |
| RUBOUT                                  | Delete the character immediately preceding the pointer (cursor) (equivalent to -D).                |
| CTRL C                                  | Return control to the monitor.   |
| ALTMODE (one only)<br>(two)             | Return control to Command Mode.<br>Direct control to Immediate Mode.                               |
| Any other character<br>than those above | Insert the character as text positioned immediately before the pointer (cursor) (equivalent to I). |



## Text Editor

### 3.8 EDIT EXAMPLE

The following example illustrates the use of some of the EDIT commands to change a program stored on the device DK. Sections of the terminal output are coded by letter and corresponding explanations follow the example.

```
A { .R EDIT
    *ERDK:TEST1.MAC$$
    *EWDK:TEST2.MAC$$
    *R$$
    */L$$
    ;TEST PROGRAM

    START:  MOV #1000,%6      ;INITIALIZE STACK
            MOV #MSG,%0      ;POINT R0 TO MESSAGE
            JSR PC,MSGTYP    ;PRINT IT
            HALT             ;STOP
B { MSG:    .ASCII/IT WORKS/
    .BYTE 15
    .BYTE 12
    .BYTE 0

C { *B 1J 5D$$
D { *GPROGRAM$$
E { *0L$$
    ;PROGRAM*I TO TEST SUBROUTINE MSGTYP. TYPES
    ;"THE TEST PROGRAM WORKS"
    ;ON THE TELETYPE\EPYTELET\TERMINAL$$
F { *F.ASCII/$$
    *8CTHE TEST PROGRAM WORKS$$
G { *P.BYTE^X
    *F.BYTE 0$V$$
    .BYTE 0

    *I
    .END
    $B/L$$
    ;PROGRAM TO TEST SUBROUTINE MSGTYP. TYPES
    ;"THE TEST PROGRAM WORKS"
    ;ON THE TERMINAL

H { START:  MOV #1000,%6      ;INITIALIZE STACK
    MOV #MSG,%0      ;POINT R0 TO MESSAGE
    JSR PC,MSGTYP    ;PRINT IT
    HALT             ;STOP
    MSG:    .ASCII/THE TEST PROGRAM WORKS/
    .BYTE 15
    .BYTE 12
    .BYTE 0
    .END

I { *EX$$
    .
```



## Text Editor

- A The EDIT program is called and prints an \*. The input file is TEST1.MAC; the output file is TEST2.MAC and the first page of input is read.
- B The buffer contents are listed.
- C Be sure the pointer is at the beginning of the buffer. Advance pointer one character (past the ;) and delete the "TEST".
- D Position pointer after PROGRAM and verify the position by listing up to the pointer.
- E Insert text. RUBOUT used to correct typing error.
- F Search for .ASCII/ and change "IT WORKS" to "THE TEST PROGRAM WORKS".
- G CTRL X typed to cancel P command. Search for ".BYTE 0" and verify location of pointer with V command.
- H Insert text. Return pointer to beginning of buffer and list entire contents of buffer.
- I Close input and output files after copying the current text buffer as well as the rest of input file into output file. EDIT returns control to the monitor.

### 3.9 EDIT ERROR MESSAGES

The Editor prints an error message whenever one of the error conditions listed next occurs. Prior to executing any commands, the Editor first scans the entire command string for errors in command format (illegal arguments, illegal combinations of commands, etc.). If an error of this type is found, an error message of the form:

?ERROR MSG?

is printed and no commands are executed. The user must retype the command.

If the command string is syntactically correct, execution is started. Execution errors are still possible, however (buffer overflow, I/O errors, etc.), and if such an error occurs, a message of the form:

?\*ERROR MSG\*?

is printed. In this case, all commands preceding the one in error are executed, while the command in error and those following are not executed. Most errors will generally be of the syntax type and can be corrected before execution.



## Text Editor

When an error occurs during execution of a Macro, the message format is:

?message IN MACRO?

or

?\*message IN MACRO\*?

depending on when it is detected.

| <u>Message</u>   | <u>Explanation</u>   |
|------------------|--|
| *CB ALMOST FULL* | The command currently being entered is within 10 characters of exceeding the space available in the Command Buffer.  |
| ?CB FULL?        | Command exceeds the space allowed for a command string in the Command Buffer.  |
| ?ILL ARG?        | The argument specified is illegal for the command used. A negative argument was specified where a positive one was expected or argument exceeds the range + or - 16,383. |
| ?ILL CMD?        | EDIT does not recognize the command specified; ED was not the first command issued when used to activate the display hardware.   |
| ?ILL MAC?        | Delimiters were improperly used, or an attempt was made to enter an M command during execution of a Macro or an EM command while an EM was in progress.                  |
| ?*DIR FULL*?     | No room in device directory for output file.   |
| ?*EOF*?          | Attempted a Read, Next or file searching command and no data was available.  |
| ?*FILE FULL*?    | Available space for an output file is full. Type a CTRL C and the CLOSE monitor command to save the data already written.  |
| ?*FILE NOT FND*? | Attempted to open a nonexistent file for editing.  |
| ?*HDW ERR*?      | A hardware error occurred during I/O. May be caused by WRITE LOCKed device. Try again.   |
| ?*ILL DEV*?      | Attempted to open a file on an illegal device, or attempted to use display hardware when none was available (it may be in use by the other job).                         |



## Text Editor

| <u>Message</u> | <u>Explanation</u>  |
|----------------|---|
| ?*ILL NAME*?   | File name specified in EB, EW, or ER is illegal.  |
| ?*NO FILE*?    | Attempted to read or write when no file is open.  |
| ?*NO ROOM*?    | Attempted to Insert, Save, Unsave, Read, Next, Change or Exchange when there was not enough room in the appropriate buffer. Delete unwanted buffers to create more room or write text to the output file. |
| ?*NO TEXT*?    | Attempted to call in text from the Save Buffer when there was no text available.  |
| ?*SRCH FAIL*?  | The text string specified in a Get, Find or Position command was not found in the available data.   |
| ?"<"ERR?       | Iteration brackets are nested too deeply or used illegally or brackets are not matched.   |



The first of these is the fact that the  
the second is the fact that the  
the third is the fact that the  
the fourth is the fact that the  
the fifth is the fact that the  
the sixth is the fact that the  
the seventh is the fact that the  
the eighth is the fact that the  
the ninth is the fact that the  
the tenth is the fact that the



## CHAPTER 4

### PERIPHERAL INTERCHANGE PROGRAM (PIP)

The Peripheral Interchange Program (PIP) is the file transfer and maintenance utility for RT-11. PIP is used to transfer files between any of the RT-11 devices (listed in Table 2-2), merge and delete files from these devices, and list, zero, and compress device directories.

#### 4.1 CALLING AND USING PIP

To call PIP from the system device type:

R PIP

in response to the dot printed by the Keyboard Monitor. The Command String Interpreter prints an asterisk at the left margin of the terminal and waits to receive a line of filenames and command switches. PIP accepts up to six input filenames and three output filenames; command switches are generally placed at the end of the command string but may follow any filename in the string. There is no limit to the number of switches which may be indicated in a command line, as long as only one operation (insertion, deletion, etc.) is represented.

Since PIP performs file transfers for all RT-11 data formats (ASCII, object, and image) there are no assumed extensions for either input or output files; all extensions, where present, must be explicitly specified.

Following completion of a PIP operation, the Command String Interpreter prints an asterisk at the left margin of the teleprinter and waits for another PIP command line. Typing CTRL C at any time returns control to the Keyboard Monitor. To restart PIP, type R PIP or the REENTER command in response to the monitor's dot.

##### 4.1.1 Using the "Wild Card" Construction

PIP follows the standard file specification syntax explained in Section 2.5 (Chapter 2) with one exception: the asterisk character can be used in a command string to represent filenames or extensions. The asterisk (called the "wild card") in a file specification means "all". For instance, "\*.MAC" means all files with the extension .MAC.



## Peripheral Interchange Program

regardless of filename. "FORTN.\*" means all files with the filename FORTN regardless of extension. "\*.\*" means all files, regardless of name or extension.

The wild card character is legal in the following cases only (switches are explained in the next section):

1. Input file specification for the copy and multiple copy operations (i.e., no switch, /I, /B, and /A).
2. File specification for the delete operation (/D).
3. Input and output file specifications for the rename operation (/R).
4. Input and output file specifications for the multiple copy operation (/X).
5. Input file specifications for the directory list operations (/L, /E, /F).

Operations on files implied by the wild card asterisk are performed in the order in which the files appear in the directory. System files with the extension .SYS and files with bad blocks and the extension .BAD are ignored when the wild card character is used unless the /Y switch is specified.

### Examples:

|                   |   |
|-------------------|---|
| **.*BAK/D         | Causes all files with the extension .BAK (regardless of their filenames) to be deleted from the device DK.  |
| **.*TST=**.*BAK/R | Renames all files with a .BAK extension (regardless of filenames) so that these files now have a .TST extension (maintaining the same filenames). |
| *RK1:**.*X/Y=**.* | Transfers all files, including system files, (regardless of filename or extension) from device DK to device RK1.                                  |
| **.*MAC,**.*OBJ/L | Lists all files with .MAC and .OBJ extensions.  |

## 4.2 PIP SWITCHES

The various operations which can be performed by PIP are summarized in Table 4-1. If no switch is specified, PIP assumes the operation is a file transfer in image (/I) mode. Detailed explanations of the switches follow the table.



# Peripheral Interchange Program

Table 4-1  
PIP Switches

| Switch          | Section | Explanation   |
|-----------------|---------|---|
| /A              | 4.2.2   | Copies file(s) in ASCII mode; ignores nulls and rubouts; converts to 7-bit ASCII.   |
| /B              | 4.2.2   | Copies files in formatted binary mode.  |
| /C              | 4.2.2   | Used in conjunction with another switch; causes only files with current date (as designated using the monitor DATE command) to be included in the specified operation.  |
| /D              | 4.2.4   | Deletes file(s) from specified device.  |
| /E              | 4.2.6   | Lists the device directory including unused spaces and their sizes. An empty space on a cassette or magtape directory represents a deleted file. Sequence numbers are listed for cassettes.   |
| /F              | 4.2.6   | Prints a short directory (filenames only) of the specified device.  |
| /G              | 4.2.2   | Ignores any input errors which occur during a file transfer and continues copying.  |
| /I or no switch | 4.2.2   | Copies file(s) in image mode (byte by byte). This is the default switch.  |
| /K              | 4.2.12  | Scans the specified device and types the absolute block numbers (in octal) of any bad blocks on the device.   |
| /L              | 4.2.6   | Lists the directory of the specified device, including the number of files, their dates, and the number of blocks used by each file. Sequence numbers are listed for cassettes.   |
| /M:n            | 4.2.1   | Used when I/O transfers involve either cassette or magtape. n represents the numeric position of the file to be accessed in relation to the physical position of the cassette or magtape on the drive. If n is positive, the tape spaces forward from its current position until either the filename or the nth file is found; if n is negative, the tape is rewound first, and then it spaces forward until either the filename or the nth file is found. If n is 0 (or not indicated) the tape is rewound and searched for the filename. For wild card operations, specification of /M with a positive argument will prevent the tape from rewinding between each file involved in the operation. |
| /N:n            | 4.2.7   | Used with /Z to specify the number of directory segments (n) to allocate to the directory.  |
| /O              | 4.2.10  | Bootstraps the specified device (DT0, RKn, or RF only).   |



# Peripheral Interchange Program

Table 4-1 (Cont.)  
PIP Switches

| Switch | Section | Explanation   |
|--------|---------|---|
| /Q     | 4.2.2   | When used in conjunction with another PIP operation, causes PIP to type each filename which is eligible for a wild card operation and to ask for a confirmation of its inclusion in the operation. Typing a "Y" causes the named file to be included in the operation; typing anything else excludes the file. The command line is not processed until the user has confirmed each file in the operation.           |
| /R     | 4.2.5   | Renames the specified file.   |
| /S     | 4.2.8   | Compresses the files on the specified directory device so that free blocks are combined into one area.  |
| /T     | 4.2.4   | Extends number of blocks allocated for a file.  |
| /U     | 4.2.9   | Copies the bootstrap from the specified file into absolute blocks 0 and 2 of the specified device.  |
| /V     | 4.2.11  | Types the version number of the PIP program being used.   |
| /W     | 4.2.6   | Includes the absolute starting block and any extra directory words in the directory listing for each file on the device (numbers in octal). Used with /F, /L, or /E.  |
| /X     | 4.2.3   | Copies files individually (without concatenation).  |
| /Y     | 4.2.2   | Causes system files and .BAD files to be operated on by the command specified. Attempted modifications or deletions of .SYS or .BAD files without /Y are not done and cause the message ?NO SYS ACTION? to be printed.  |
| /Z:n   | 4.2.7   | Zeroes (initializes) the directory of the specified device; n is used to allocate extra words per directory entry. When used with /N, the number of directory segments for entries may be specified. When used with cassette, /Z writes a sentinel file at the beginning of the tape; with magtape, /Z writes a volume label followed by a dummy file followed by double tape marks indicating logical end-of-tape. |

## 4.2.1 Operations Involving Magtape or Cassette

PIP operations involving cassette and magtape devices are handled somewhat differently than other RT-11 devices, because of the sequential nature of these devices. The last file on a cassette or magtape (the logical end-of-tape) is specially formatted so that it marks the end of current data and indicates where new data may begin (double end-of-file for magtape, sentinel file or physical end-of-tape for cassette). Therefore, operations which designate specific block lengths (such as /T and /N) are meaningless, and unused spaces on the tape (resulting from file deletions) cannot be filled.



## Peripheral Interchange Program

PIP operations which are legal using cassette and magtape include the following: /A, /B, /D, /E, /F, /G, /I, /L, /M, /Q, /V, /W, /X, /Y, and /Z. Usually the device (CT or MT) is rewound each time an operation is performed. Since there is no inclusive directory at the beginning of the tape the only way to access a file is to search the tape from the beginning until it is found. However, the /M:n switch is available for situations where it is not necessary or desirable to rewind the tape before each operation. If the argument (n) is positive, the operation indicated will not rewind the tape first, but will space forward until it finds either the nth file, the filename indicated in the command line, or the logical end-of-tape, whichever occurs first. If the argument is negative, the cassette or magtape will be rewound first and then spaced forward until the filename (or nth file, or logical end-of-tape) is found. Thus:

/M:1                means suppress rewind, begin operation at current position.

/M:-1              means rewind tape and access the first file on it.

Remember that when /M:n is used, n is interpreted as an octal number. /M:n must be used if it is intended that n represent a decimal number.

For example, assume the directory of a cassette on unit 1 is:

```
17-JUL-74
FILE .1      0  5-MAY-74
FILE .2      0  5-MAY-74
FILE .3      1 13-MAY-74
FILE .4      1 28-JUN-74
FILE .5      0 17-JUL-74
5 FILES, 2 BLOCKS
*
```

and the last PIP operation involved FILE.4, leaving the cassette positioned at the end of FILE.4. To access FILE.2, the next operation (for example, deleting FILE.2) could use the /M construction:

```
*CT1:DUM/M:-2/D
```

In this case, the cassette rewinds first, then spaces forward from its current position to the second file in sequence and deletes it. (In a delete operation, the dummy filename is necessary; otherwise, a non-file structured delete is performed and the tape is zeroed. See Section 4.2.4).

Another useful application of the /M switch involves a case where a number of files are to be created on a magtape or cassette. Using the construction:

```
*MT:*. */X=FILE.1.FILE.2.../M:1000
```

prevents a rewind from occurring before each new file is created on the tape. Normal operation (when creating a new file on magtape or cassette) is to rewind, then search the tape for the logical end. If a file with the same name as the one being created is encountered, it is deleted and the new file is opened at the logical end of the tape. The /M:1000 command first causes the tape to space forward until it reaches the logical end-of-tape, (assuming less than 1000 (octal) files on the tape), at which point the next file is entered, and so on. If the tape were already positioned at the end of the tape, an



## Peripheral Interchange Program

/M:1 would suffice to cause the new file to be written there. Note that creation of a new file with the /M switch can result in several files with the same name on the same tape; those files occurring before the tape position are not searched for duplication prior to the creation of the new file.

RT-11 magtapes sometimes contain a dummy file at the beginning of the tape, which is written when the tape is initialized with the /Z switch. This file shows up in extended directories (/E) as an <UNUSED> entry in the first file position. Deleted files on magtape or cassette do not show up in /F or /L directory listings, but must always be considered when the /M:n switch is used. Care must always be taken to use a /E directory when counting file position prior to using that position as an /M:n argument; <UNUSED> files must be counted as files on the tape.

For example:

```
. R PIP
*MT0:/E
11-SEP-74
< UNUSED >      0
A      .MAC      40 11-SEP-74
B      .MAC      15 11-SEP-74
< UNUSED >      2
D      .MAC       2 11-SEP-74
3 FILES, 57 BLOCKS
```

Extended directory: shows  
absolute file positions.

```
*MT0:/L
11-SEP-74
A      .MAC      40 11-SEP-74
B      .MAC      15 11-SEP-74
D      .MAC       2 11-SEP-74
3 FILES, 57 BLOCKS
```

Normal directory; does  
not accurately display  
file positions.

If the user wished to access file A.MAC on the magtape in the example above, /M:-2 must be used (/M:-1 would access the first empty file). Likewise, B.MAC is accessed with /M:-3. Rewind can also be suppressed for cassette and magtape as input devices by specifying a very large number in conjunction with wild card transfers from magtape or cassette.

```
**. *=MT0:*. */M:2000
?OUT FIL?
```

This transfers all files from MT0: to DK: without rewinding between each file. The argument 2000 is an arbitrarily large number; any number larger than the actual number of files on the tape will suffice. An error message indicates when all files have been transferred.

The most common method for spacing to the end of the tape is:

```
*DUMMY=MT0:DUMMY/M:2000
?FIL NOT FND?
```

where DUMMY is a file name which does not exist on the tape. Note that an error message is printed when the end of the tape is reached.



## Peripheral Interchange Program

Directory listings of magtapes include the length of each file in 256(decimal) word blocks. In cassette directories, however, sequence numbers rather than block numbers are printed. Sequence numbers indicate the sequential ordering of a file in cases where it has been continued on more than one cassette. In the example cassette directory listing (at the beginning of this section), the numbers in the middle column represent sequence numbers; both FILE.3 and FILE.4 are the second segments of continued files. All files on cassette are initially assigned a sequence number of 0. The sequence number is automatically updated whenever the file must be continued as a result of a full cassette.

During I/O transfer operations involving cassette, if the cassette is full before the transfer has finished, the message:

CTn: PUSH REWIND OR MOUNT NEW VOLUME

is printed; n represents the number of the drive (0 or 1) on which the current cassette is mounted. If the cassette rewind button is subsequently pushed, an error message is typed (IN or OUT ERR) and the tape is rewound.

To continue an output operation, mount a new cassette (which has been properly formatted as described in Section 4.2.7) on the same drive. The new cassette is rewound automatically and a file is opened on it under the same name and extension; the sequence number in its directory is updated to reflect the continuation, and the transfer continues.

If the message occurs during an input operation, mount the cassette containing the continued portion of the file on the drive; the cassette is rewound first. PIP then looks for a file with the same name and extension and the proper sequence number and continues the input operation. The message is repeated if the next segment is not found.

For example:

```
*CT0:FILE.AGA=DT1:ASC.MAC,DK:BALOR.MAC/A
CT0: PUSH REWIND OR MOUNT NEW VOLUME
```

This copies in ASCII mode the file ASC.MAC from DECTape 1 and BALOR.MAC from device DK and combines them under the name FILE.AGA on CT0. The cassette runs out of room and requests that a new one be mounted. The operation continues automatically when the second cassette has been mounted.

A directory of the second cassette in the above operation is next requested; note that the sequence number of FILE.AGA is 1, signifying it is the second part of a continued file.

```
*CT0:/L
23-MAY-74
TRA .BIN 0 16-FEB-74
FILE .AGA 1 23-MAY-74
2 FILES, 1 BLOCKS
*
```

(The number of blocks in a cassette directory simply represents the total of sequence numbers in the directory.)

Any cassette mounted in response to a continuation message MUST have been previously initialized at some time as described in Section 4.2.7.



## Peripheral Interchange Program

If a full cassette is mounted or an attempt is made to access some file on it that does not exist, the continuation message re-occurs. The operation may be continued by mounting another cassette.

Note that if an attempt is made to access a file which has a non-zero sequence number (during some operation which is not a continuation of an operation), the file will not be found.

If the end of a tape is reached during a magtape I/O operation, an IN or OUT ERR message is printed. Continuation on another magtape is not allowed and the operation must be repeated using a different magtape.

If CTRL C is typed during any output operation to cassette or magtape, an end-of-tape or sentinel file is not written on the tape first. Consequently, no future enters may occur to the tape unless one of two recovery procedures is followed:

1. Transfer all good files from the bad tape to another tape and zero the bad tape in the following manner:

```
*dev1:*,*/X=dev0:file1,file2,...fileN/M:1000
*dev0:/Z
dev0:/Z ARE YOU SURE ?
```

This causes a logical end-of-tape to be written onto the bad tape and makes it again available for use.

2. Determine the sequential number of the file which was interrupted and use the /M construction to enter a replacement file (either a new file or a dummy file). Assuming the bad file is the 4th file on the tape, use a command line of this construction:

```
*dev0:file.new=file.dum/M:-4
```

A logical end-of-tape now exists on the tape, making it available for use.

Since magtapes and cassettes are not random access devices, each unit can have only one file accessed at a time. Avoid PIP command strings which specify the same unit number for both input and output, since a loss of information can occur. For example:

```
*CT0:FILE1.MAC=CT0:FILE1.MAC
?FIL NOT FND?
*
```

The result of this operation is to delete FILE1.MAC before the error message is printed. The filename is not deleted from the directory, however.

Recovery procedures for errors caused by bad tapes are described in RT-11 SOFTWARE SUPPORT MANUAL (DEC-11-ORPGA-B-D).



## Peripheral Interchange Program

### 4.2.2 Copy Operations

A command line without a switch causes files to be copied onto the destination device in image mode (byte by byte). This operation is used to transfer memory image (save format) files and any files other than ASCII or formatted binary. For example:

\*ABC<XYZ                      Makes a copy of the file named XYZ on device DK and assigns the name ABC. (Both files exist on device DK following the operation).

\*SY:BACK.BIN=PR:/I          Copies a tape from the papertape reader to the system device in image mode and assigns it the name BACK.BIN.

The /A switch is used to copy file(s) in ASCII mode as follows:

\*DT1:F1<F2/A                Copies F2 from device DK onto device DT1 in ASCII mode and assigns the name F1.

Nulls and rubouts are ignored in an ASCII mode file transfer.

The /B switch is used to transfer formatted binary files. The formatted binary copy switch should be used for .OBJ files produced by the assembler or FORTRAN and for .LDA files produced by the Linker. For example:

\*DK:FILE.OBJ<PR:/B          Transfers a formatted binary file from the papertape reader to device DK and assigns the name FILE.OBJ.

When performing formatted binary transfers, PIP verifies checksums and prints the message ?CHK SUM? if a checksum error occurs.

To combine more than one file into a single file, use the following format:

\*DK:AA<DT1:BB,CC,DD/I        Transfers files BB, CC and DD to device DK as one file and assigns this file the name AA.

\*DT3:MERGE=DT2:FILE2,FILE3/A      Merges ASCII files FILE2 and FILE3 on DT2 into one ASCII file named MERGE on device DT3.

Errors which occur during the copy operation (such as a parity error) cause PIP to output an error message and return for another command string.

The /G switch is used to copy files but ignore all input errors. For example:

\*ABC<DT1:TOP/G                Copies file TOP in image mode from device DT1 to device DK and assigns the name ABC. Any errors during the copy operation are ignored.



## Peripheral Interchange Program

\*DT2:COMB<DT1:F1,F2/A/G

Copies files F1 and F2 in ASCII mode from device DT1 to device DT2 as one file with the name COMB. Ignores input errors.

The wild card construction may be used for input file specifications during copy operations. Be sure to use the /Y switch if system files (.SYS) are to be copied. For example:

\*DT1:PROG1<\*.MAC

Copies, in image mode, all files with a .MAC extension from device DK to device DT1 and combines them under the name PROG1.

\*\*.\*=DT3:\*.\*/G/Y/X

Copies to device DK, in image mode, all files (including .SYS files) from device DT3; ignores any input errors.

If only files with the current date are to be copied (using the wild card construction), the /C switch must also be used in the command line. For example:

\*DT2:NN3=ITEM1.\*/C.ITEM2/A

Copies, in ASCII mode, all files having the filename ITEM1 and the current date, (the date entered using the monitor DATE command) and copies ITEM2 (regardless of its date) from device DK to device DT2 and combines them under the name NN3.

\*DT3:\*. \*\*.\* \*/C/X

Copies all files with the current date from DK to DT3. Note that commands of this nature are an efficient way to backup all new files after a session at the computer.

The /Q switch is used in conjunction with another PIP operation and the wild card construction to list all files and allow the user the opportunity to confirm individually which of these files should be processed during the wild card expansion. Typing a "Y" causes the named file to be processed; typing anything else excludes the file. For example:

\*\*.OBJ<DT1:\*.OBJ/Q/X

FIRST .OBJ?Y

GETR .OBJ?

BORD .OBJ?

CARJ .OBJ?Y

Copies the files FIRST.OBJ and CARJ.OBJ to the disk in image mode from DECTape 1 and ignores the others.

The file allocation scheme for RT-11 normally allows half the entire largest available space or the second largest space, or a maximum size (a constant which may be patched in the RT-11 monitor; see GETTING STARTED WITH RT-11, DEC-11-ORCPA-D-D), whichever is largest, for a new file. The user can, using the [n] construction explained in Chapter 2, force RT-11 to allow the entire largest possible space by setting n=177777. If n is set equal to any other value (other than 0 which is default and gives the normal allocation described first above), that size will be allocated for the file.



## Peripheral Interchange Program

Therefore, assume that the directory for a given device shows a free area of 200 blocks and that PIP returns an ?OUT ER? message when a transfer is attempted to that device with a file which is longer than 100 blocks but less than 200 blocks. Transfers in this situation can be accomplished in either of two ways:

1. Use the [n] construction on the output file to specify the desired length (refer to Chapter 2, Section 2.5 for an explanation of the [n] construction).
2. Use the /X switch during the transfer to force PIP to allocate the correct number of blocks for the output file. This procedure will operate correctly if the input device is DECTape or disk.

For example, assume that file A is 150 blocks long and that a directory listing shows that there is a 200 block <unused> space on DT1:

```
.R PIP
*DT1:A=A
?OUT ER?                               File longer than 100 blocks.
```

```
or
*DT1:A[150]=A
*DT1:A=A/X                             Either command causes a correct
                                         transfer.
```

### 4.2.3 Multiple Copy Operations

The /X switch allows the transfer of several files at a time onto the destination device as individual files. The /A, /G, /C, /Q, /B and /Y switches can be used with /X. If /X is not indicated, all output files but the first will be ignored.

#### Examples:

```
*FILE1,FILE2,FILE3<DT1:FILEA,FILEB,FILEC/X
Copies, in image mode, FILEA, FILEB and
FILEC from device DT1 to device DK as
separate files called FILE1, FILE2 and
FILE3, respectively.
```

```
*DT2:F1.*=F2.* /X
?NO SYS ACTION?
*
Copies, in image mode, all files named
F2 (except files with .SYS or .BAD
extensions) from device DK to device
DT2. Each file is assigned the filename
F1 but retains its original extension.
```

```
*DT1:*.*=DT2:*.*/X
?NO SYS ACTION?
Copies, in image mode, all files on
device DT2 to device DT1 (except files
with .SYS or .BAD extensions); the files
are copied separately and retain the
same names and extensions.
```

```
*DT1:FILE1,FILE2<FILEA.* /A/G/X
This command line assumes there are two
files with the filename FILEA (and any
extension excluding .SYS or .BAD
extensions) and copies these files in
```



## Peripheral Interchange Program

ASCII mode to device DT1. The files are transferred in the order they are found in the directory; the first file found is copied and assigned the name FILE1, and the second is assigned FILE2. If there is a third, it is ignored and a fourth causes an ?OUT FIL? error.

```
*DT0:*.SYS=*.SYS/X/Y
```

Copies all system files from device DK to device DT0.

File transfers performed via normal operations place the new file in the largest available area on the disk. The /X switch, however, places the copied files in the first free place large enough to accommodate it. Therefore, the /X switch should be used whenever possible (i.e., when no concatenation is desired) as an aid to reducing disk fragmentation.

```
*A=B
```

and

```
*A=B/X
```

perform the same operation; however, using the second construction whenever possible increases the system disk-usage efficiency.

For example, assume the directory of DT1 is:

```
9-MAY-74
MONITR.SYS 32 5-MAY-74
< UNUSED > 2
PR .SYS 2 5-MAY-74
< UNUSED > 528
2 FILES, 34 BLOCKS
530 FREE BLOCKS
```

To copy the file PP.SYS (2 blocks long) from DK to DT1, the command:

```
*DT1:PP.SYS=PP.SYS/Y
```

can be entered, and the new directory is:

```
9-MAY-74
MONITR.SYS 32 5-MAY-74
< UNUSED > 2
PR .SYS 2 5-MAY-74
PP .SYS 2 9-MAY-74
< UNUSED > 526
3 FILES, 36 BLOCKS
528 FREE BLOCKS
```

If the command:

```
*DT1:PP.SYS=PP.SYS/Y/X
```

had been entered, the new directory would appear:



## Peripheral Interchange Program

```
9-MAY-74
MONITR.SYS 32 5-MAY-74
PP .SYS 2 9-MAY-74
PR .SYS 2 5-MAY-74
< UNUSED > 528
3 FILES, 36 BLOCKS
528 FREE BLOCKS
```

### 4.2.4 The Extend and Delete Operations

The /T switch is used to increase the number of blocks allocated for the specified file. The file associated with the /T switch must be followed by a numeric argument of the form [n] where n is a decimal number indicating the number of blocks to be allocated to the file at the completion of the extend operation.

The format of the /T switch is:

```
dev:filnam.ext[n]=/T
```

A file can be extended in this manner only if it is followed by an unused area of sufficient size (on whichever device it is located) to accommodate the additional length of the extended file. It may be necessary to create this space by moving other files on the device using the /X switch.

Specifying the /T switch in conjunction with a file that does not currently exist creates a file of the designated length.

Error messages are printed if the /T command makes the specified file smaller (?EXT NEG?) or if there is insufficient space following the file (?ROOM?).

#### Examples:

```
*ABC[200]=/T      Assigns 200 blocks to file ABC on device
                   DK.
*DT1:XYZ[100]</T   Assigns 100 blocks to the file named XYZ
                   on device DT1.
```

The /D switch is used to delete one or more files from the specified device. The wild card character (\*) can be used in conjunction with this command.

Only six files can be specified in a delete operation if each file to be deleted is individually named (i.e., if the wild card character is not used).

A cassette or magtape may be initialized by indicating the /D switch and omitting any filenames. For example:

```
*MT:/D
*CT:/D
```

Both devices are zeroed. This is not the case with the other RT-11 devices, where omission of a filename causes no action to occur.



## Peripheral Interchange Program

When a file is deleted on block-replaceable devices, the information is not destroyed. The file name is merely removed from the directory. If a file has been deleted but not overwritten, it can be recovered with the /T switch by specifying a command of the form:

```
filena.ext[n]=/T
```

where filena.ext is the name desired and n is the length of the deleted file. For example:

```
*DT1:/E
4-JUN-74
A      .MAC      18  3-JUN-74
B      .MAC      17  3-JUN-74
C      .MAC      19  3-JUN-74
< UNUSED > 510
3 FILES, 54 BLOCKS
510 FREE BLOCKS
```

```
*DT1:B.MAC/D
```

```
*DT1:/E
4-JUN-74
A      .MAC      18  3-JUN-74
< UNUSED > 17
C      .MAC      19  3-JUN-74
< UNUSED > 510
2 FILES, 37 BLOCKS
527 FREE BLOCKS
```

File B.MAC could now be recovered by:

```
*DT1:B.MAC[17]=/T
```

The /T switch looks for the first unused area large enough to accommodate the requested file length. If the file to be recovered is in the first area large enough to accommodate the size specified, the preceding command is sufficient. If not, all larger unused spaces preceding the desired file must be given dummy names before the recovery can be made.

For instance, assume the previous example with the exception that A.MAC has a 33 block unused file before it, so that the directory looks like:

```
*DT1:/E
4-JUN-74
< UNUSED > 33
A      .MAC      18  3-JUN-74
< UNUSED > 17
C      .MAC      19  3-JUN-74
< UNUSED > 477
2 FILES, 37 BLOCKS
527 FREE BLOCKS
```

A recovery of B.MAC would require:

```
*DT1:DUMMY[33]=/T
*DT1:B.MAC[17]=/T
```



## Peripheral Interchange Program

If the 33 block unused area was not named prior to B.MAC, the first 17 blocks of the 33 block area would become B.MAC. Note that magtape and cassette files cannot be recovered once deleted.

### Examples:

|                         |   |
|-------------------------|---|
| *FILE1.SAV/D            | Deletes FILE1.SAV from device DK.   |
| *DT1:*.*/D              | Deletes all files from device DT1 except those with a .SYS or .BAD extension. If there is a file with a .SYS or .BAD extension, the message ?NO SYS ACTION? is printed to remind the user that these files have not been deleted. |
| ** .MAC/D               | Deletes all files with a .MAC extension from device DK.   |
| *DT1:B1,DT2:R1,DT3:AA/D | Deletes the files specified from the associated devices.  |
| *RK1:*.*/D/Y            | Deletes all files from device RK1.  |

### 4.2.5 The Rename Operation

The /R switch is used (in a manner similar to the multiple copy command described in Section 4.2.3) to rename a file given as input with the associated name given in the output specification. There must be an equal number of input and output files and they must reside on the same device, or an error message will be printed. The /Y switch must be used in conjunction with /R if .SYS files are to be renamed.

The Rename command is particularly useful when a file on disk or DECTape contains bad blocks. By renaming the file with a .BAD extension, the file permanently resides in that area of the device so that no other attempts to use the bad area will occur. Once a file is given a .BAD extension it cannot be moved during a compress operation. .BAD files are not renamed in wild card operations unless /Y is used.

### Examples:

|                        |  |
|------------------------|--|
| *DT1:F1,X1<DT1:F0,X0/R | Renames F0 to F1 and X0 to X1 on device DT1.   |
| *FILE1.*<FILE2.* /R    | Renames all files on device DK with the name FILE2 (except files with .SYS or .BAD extension) to FILE1, retaining the original extensions. |

### 4.2.6 Directory List Operations

The /L switch lists the directory of the specified device. The listing contains the current date, all files with their associated creation dates, total free blocks on the device if disk or DECTape, the number of files listed, and number of blocks used by the files



## Peripheral Interchange Program

(sequence number for cassette). File lengths, number of blocks and number of files are indicated as decimal values. If no output device is specified, the directory is output to the terminal (TT:).

### Examples:

```
*DT1:/L
  1-AUG-74
MONITR.SYS    32  5-MAY-74
PP    .SYS    2   9-MAY-74
PR    .SYS    2   5-MAY-74
F2    .REL    15
MERGE                2
COMB                2
6 FILES, 55 BLOCKS
509 FREE BLOCKS
```

Outputs complete directory of device DT1 to the terminal.

```
*DIRECT=DT3:/L
```

Outputs complete directory of device DT3 to a file, DIRECT, on the device DK.

```
** .MAC/L
  1-AUG-74
VTMAC .MAC    7 22-JUL-74
FILE2 .MAC    1
2 FILES, 8 BLOCKS
3728 FREE BLOCKS
*
```

Lists on the terminal a directory of files on device DK with the extension .MAC.

```
*CT1:/L
10-SEP-74
PAT1 .FOR    0 10-SEP-74
PAT2 .FOR    0 10-SEP-74
IMUL .OBJ    0 10-SEP-74
SQRT .FTN    0 10-SEP-74
4 FILES, 0 BLOCKS
```

Lists a directory of all files on cassette drive 1.

The /E switch lists the entire directory including the unused areas and their sizes in blocks (decimal); an empty space appears in cassette and magtape directories to designate a deleted file.

### Examples:

```
*/E
  9-SEP-74
BATCH .HLP    2 23-AUG-74
CHESS .SAV   20 23-AUG-74
PAT1 .FOR   10 23-AUG-74
IRAD50.MAC    8 23-AUG-74
.
```

Outputs to the terminal a complete directory of the device DK including the size of unused areas.



# Peripheral Interchange Program

```
< UNUSED >      2
TRIG .OBJ        2  6-SEP-74

STP  .OBJ        2  6-SEP-74
BAC  .OBJ        2  6-SEP-74
< UNUSED >      20
```

```
LIBR1 .OBJ      137  6-SEP-74
DIRECT      1  9-SEP-74
< UNUSED >    230
254 FILES, 4280 BLOCKS
498 FREE BLOCKS
```

```
*LP:=CT1:/E
11-SEP-74
A      .MAC      0 11-SEP-74
A      .MAC      0 11-SEP-74
B      .MAC      0 11-SEP-74
3 FILES, 0 BLOCKS
```

Outputs to the line printer  
a complete directory of  
cassette drive 1. 0's  
represent segment numbers.

The /F switch lists only filenames, omitting the file lengths and associated dates.

## Examples:

```
*DT0:/F
TRACE .MAC
CARGO .REL
BMAP .OBJ
AAA
```

Outputs a filename directory  
of the device DT0 to the  
terminal.

```
*LP:=CT1:/F
```

Outputs a filename directory  
of the device CT1 to the line  
printer.

```
A      .MAC
A      .MAC
B      .MAC
```

The /L, /E and /F commands have no effect on the files of the specified device. If the /W switch is used in conjunction with the /F, /L, or /E switches, the absolute starting block of the file and extra words (in octal) will be included in the listing (for all but cassette and magtape). For example:

```
*RK1:/L/W
10-SEP-74
DSQRT .OBJ      1 10-SEP-74      16      0
MAIN  .OBJ      1 10-SEP-74      17      0
BASICK.OBJ     11 10-SEP-74      20      0
OTSV2 .OBJ      3 10-SEP-74      33      0
```

The first three columns indicate the filename and extension, block length, and date. The fourth column shows the absolute starting block (in octal), and the fifth column shows the contents of each extra word per directory entry (in octal). (This is allocated using the /Z:n switch; see Section 4.2.7.)



## Peripheral Interchange Program

Using the /L, /E, or /F switch in conjunction with a device and filename causes the filename, and optionally the date and file length, to be output rather than a directory of the entire device. For example:

```
*F1.SAV/L
```

causes:

```
4-JUN-74
F1 .SAV 18 4-JUN-74
3710 FREE BLOCKS
*
```

to be output, providing the file exists on device DK.

Directories are made up of segments which are two blocks long. Full directory listings with multiple segments contain blank lines as segment boundaries.

### 4.2.7 The Directory Initialization Operation

The /Z switch clears and initializes the directory on the specified device (writes a logical end-of-tape file to cassette and magtape). /Z must always be used to create an empty file directory before using a volume for the first time.

The form of the switch is:

```
/Z:n
```

where n is an octal number specifying the number of extra words per directory entry. If n is not used, no extra words are allocated and 70 entries can be made in a directory block. When extra words are allocated, the formula for determining the number of entries per directory block is:

$$507/((\# \text{ of extra words})+7)$$

For example, if the switch /Z:1 is used, 63 entries can be made per block.

When /Z is used, PIP responds as follows:

```
device/Z ARE YOU SURE ?
```

For example:

```
*DT1:/Z
DT1:/Z ARE YOU SURE ?
```

Answer Y and a carriage return to perform the initialization. An answer beginning with a character other than Y is considered to be no.

Example:

```
*DT1:/Z
DT1:/Z ARE YOU SURE ?Y<CR>
*
Zeroes the directory on device DT1 and
allocates no extra words for the
directory.
```



## Peripheral Interchange Program

The /N switch is used with /Z to specify the number of directory segments for entries in the directory. The form of the switch is:

/N:n

where n is an octal number. If n is not specified, four segments are allocated. The maximum number of segments which can be allocated is 37(8).

Example:

\*RK1:/Z:2/N:6

Zeroes the directory on device RK1, allocates two extra words per directory entry and allocates six directory segments for entries.

### 4.2.8 The Compress Operation

The /S switch is used to compress the directory and files on the specified device, condensing all the free (unused) blocks into one area.

Input errors are reported on the console terminal unless the /G switch is used; output errors are always reported. In either case, the compress continues. /S can also be used to copy DECTapes and disks (/S will not copy the bootstrap file in absolute blocks 0 and 2). It is important to note that the /S switch destroys any previous directory on the output device. The new directory on the output device will have the same number of segments as the directory on the input device.

To increase the number of directory blocks in a two-volume compress (that is, from one volume to another rather than from one volume to itself), use the /N:n switch in conjunction with the /S switch (any attempts to decrease the directory size are ignored).

/S does not move files with the .BAD extension. This feature provides protection against reusing bad blocks which may occur on a disk. Files containing bad blocks can be renamed with the .BAD extension and are then left in place when a /S is executed.

If a compress operation is performed on the system device, the message:

?REBOOT?

is printed to indicate that it may be necessary to reboot the system. If .SYS files were not moved during the compress operation, it is not necessary to reboot the system.

### NOTE

Rebooting the system in response to the ?REBOOT? warning message should ONLY be done AFTER the operation which generated the message is complete. ?REBOOT? does not signify that the system should be



## Peripheral Interchange Program

rebooted immediately; the user should wait for the "\*" signifying that PIP is ready for another command before rebooting.

If the command attempts to compress a large device to a smaller one, an error results and the directory of the smaller device is zeroed. If a device is being compressed in place, input and output errors are reported on the terminal and the operation continues to completion.

### Examples:

|               |  |
|---------------|--|
| *SY:/S        | Compresses the files on the system   |
| ?REBOOT?      | device SY:   |
|               |  |
| *DT1:A<DT2:/S | Transfers and compresses the files from device DT2 to device DT1. Device DT2 is not changed. The filename A is a dummy specification required by the Command String Interpreter. |

/S cannot be used when a foreground job is present; a ?FG PRESENT? error message results if this is attempted.

### 4.2.9 The Bootstrap Copy Operation

The bootstrap copy switch (/U) copies the bootstrap portion of the specified file into absolute blocks 0 and 2 of the specified device.

### Examples:

|                         |  |
|-------------------------|--|
| *DK:A<DK:MONITR.SYS/U   | Writes the bootstrap file MONITR.SYS in blocks 0 and 2 of the device DK. A is a dummy filename.  |
|                         |  |
| *DT0:F2=DK:MONITR.SYS/U | Writes the bootstrap file MONITR.SYS into blocks 0 and 2 of the device DT0. The filename F2 is a dummy specification required by the Command String Interpreter. |

### 4.2.10 The Boot Operation

The boot switch reboots the system, reinitializing monitor tables and returning the system to the monitor level. The boot switch performs the same operation as a hardware bootstrap.

### Example:

|        |                        |
|--------|------------------------|
| *DK:/O | Reboots the device DK. |
|--------|------------------------|



## Peripheral Interchange Program

If a boot switch is specified on a non-file structured device, the message:

?BAD BOOT?

is printed. Legal system devices are DT0, RK0-RK7, RF, SY, and DK. Note that /O is legal if a foreground job is present; the ?FG PRESENT? error message results.

### 4.2.11 The Version Switch

The Version switch (/V) outputs a version number message (representing the version of PIP in use) to the terminal using the form:

PIP V02-XX

The rest of the command line, if any, is ignored.

### 4.2.12 Bad Block Scan (/K)

The bad block switch (/K) scans the specified device and types the absolute block numbers of those blocks on the device which return hardware errors. The block numbers typed are octal; the first block on a device is 0(8). Note that if no errors occur, nothing will be output. A complete scan of a disk pack takes several minutes.

Example:

|                  |                                   |
|------------------|-----------------------------------|
| *RK2:/K          | Scan disk drive 2 for bad blocks. |
| BLOCK 140 IS BAD |                                   |
|                  |                                   |
| *RK1:/K          | Scan drive 1. No blocks are bad.  |
| *                |                                   |

#### 4.2.12.1 Recovery from Bad Blocks

As a disk ages, the recording surface wears. Eventually unrecoverable I/O errors occur during attempts to read or write a bad disk block. PIP protects against usage of bad disk areas by ignoring files with a .BAD extension (unless the /Y switch is used). Once a bad block is uncovered in an I/O operation, it can be located using the /K switch and a .BAD file can be created which encompasses the bad block.

When a hardware I/O error is detected, the recovery procedure is as follows:

1. Use the PIP /K switch to scan the device and print on the terminal the absolute block numbers (in octal) of the bad blocks. For example:

```
. R PIP
*RK1:/K
BLOCK 7723 IS BAD
*
```



# Peripheral Interchange Program

2. Obtain an extended directory with the /W switch, showing the starting block numbers of all the files on the disk.
3. If a bad block occurs in a file with valuable information, copy the file to another file using the /G switch. In most cases, only 1 bit (character) of the file is affected.
4. If the file is small, it can then be renamed with a .BAD extension to prevent further use of that disk area.
5. If the file is large or the bad block occurs in an empty area, a 1-block .BAD file can be created using the /T switch as follows:
  - a. Delete the bad file (if any).
  - b. If the bad block is at block n of the free area, create a file of length n-1 with the /T switch. Remember that there must be no spaces larger than n-1 blocks before the desired one (refer to Section 4.2.4). Also note that the block numbers printed in the /K and /W operations are octal, while the argument to the /T operation is decimal.
  - c. Create a 1-block .BAD file with the /T switch to cover the bad block.
  - d. Delete any temporary files created during the operation.

For example, assume the extended directory is:

```

.
.
NEWSRC.BAT      8 11-SEP-74    6203
RTTEMP.BAT     27 11-SEP-74    6213
PIP .MAC       150 12-SEP-74    6246
< UNUSED >     154
VERIFY.SAV      3              6726
< UNUSED >     300
PIP .OBJ        15 12-SEP-74    7405
MKPIP .CTL       1 12-SEP-74    7424
MKV2RK.CTL       4 12-SEP-74    7425
VTLIB .OBJ       10 12-SEP-74    7431
< UNUSED >     150
A                4 12-SEP-74    7671
PIP .LST       300  3-SEP-74    7675  Block 7723 (octal) of
.                                     PIP.LST is bad.
.

```

and a bad block is detected at block 7723 (octal) of the file PIP.LST. To recover, make a copy, ignoring the error, and delete the bad file:

```

*RK1:PIPR.LST=RK1:PIP.LST/G
*RK1:PIP.LST/D

```

The directory now reads:

```

.
.
NEWSRC.BAT      8 11-SEP-74    6203
RTTEMP.BAT     27 11-SEP-74    6213
PIP .MAC       150 12-SEP-74    6246

```



# Peripheral Interchange Program

```

< UNUSED > 154
VERIFY.SAV 3 6726
PIPA .LST 300 18-SEP-74 6731
PIP .OBJ 15 12-SEP-74 7405
MKPIP .CTL 1 12-SEP-74 7424
MKV2RK.CTL 4 12-SEP-74 7425
VTLIB .OBJ 10 12-SEP-74 7431
< UNUSED > 150
A 4 12-SEP-74 7671
:

```

An unused area following A contains block 7723 (octal), which is bad. Continuing in PIP:

```

*RK1:TEMP.002[154]=/T
*RK1:TEMP.003[150]=/T
*RK1:TEMP.004[22]=/T

```

This fills the unused areas with temporary files. Specifying TEMP.004 with a length of 22 blocks makes the file just long enough to precede the bad block (i.e., 7675 (octal) and 20 (decimal) equal 7723, which would be the starting block number of the next file created). The directory now contains:

```

:
NEWSRC.BAT 8 11-SEP-74 6203
RTTEMP.BAT 27 11-SEP-74 6213
PIP .MAC 150 12-SEP-74 6246
TEMP .002 154 18-SEP-74 6474
VERIFY.SAV 3 6726
PIPA .LST 300 18-SEP-74 6731
PIP .OBJ 15 12-SEP-74 7405
MKPIP .CTL 1 12-SEP-74 7424
MKV2RK.CTL 4 12-SEP-74 7425
VTLIB .OBJ 10 12-SEP-74 7431
TEMP .003 150 18-SEP-74 7443
A 4 12-SEP-74 7671
TEMP .004 22 18-SEP-74 7675
:

```

Continuing with PIP:

```
*RK1:FILE.BAD[1]=/Y/T
```

Create a bad file.

The directory now contains:

```

:
NEWSRC.BAT 8 11-SEP-74 6203
RTTEMP.BAT 27 11-SEP-74 6213
PIP .MAC 150 12-SEP-74 6246
TEMP .002 154 18-SEP-74 6474
VERIFY.SAV 3 6726
PIPA .LST 300 18-SEP-74 6731
PIP .OBJ 15 12-SEP-74 7405
MKPIP .CTL 1 12-SEP-74 7424
MKV2RK.CTL 4 12-SEP-74 7425
VTLIB .OBJ 10 12-SEP-74 7431
TEMP .003 150 18-SEP-74 7443
A 4 12-SEP-74 7671

```



## Peripheral Interchange Program

```
TEMP .004    22 18-SEP-74    7675
FILE .BAD     1 18-SEP-74    7723
```

Bad block is here.

Next delete all temporary files and rename PIPA.LST to PIP.LST. The final directory now contains:

```
NEWSRC.BAT    8 11-SEP-74    6203
RTTEMP.BAT   27 11-SEP-74    6213
PIP .MAC    150 12-SEP-74    6246
< UNUSED >   154
VERIFY.SAV     3                6726
PIP .LST    300 18-SEP-74    6731
PIP .OBJ     15 12-SEP-74    7405
MKPIP .CTL     1 12-SEP-74    7424
MKV2RK.CTL     4 12-SEP-74    7425
VTLIB .OBJ    10 12-SEP-74    7431
< UNUSED >   150
A              4 12-SEP-74    7671
< UNUSED >   22
FILE .BAD     1 18-SEP-74    7723
```

Disks with many bad blocks can often be reused by reformatting them. First copy all desired files, since reformatting destroys all information contained on a volume.

### 4.3 PIP ERROR MESSAGES

The following error messages are output on the terminal when PIP is used incorrectly:

| <u>Errors</u> | <u>Meaning</u>  |
|---------------|---|
| ?BOOT COPY?   | An error occurred during an attempt to write bootstrap with /U switch.  |
| ?CHK SUM?     | A checksum error occurred in a formatted binary transfer.   |
| ?COR OVR?     | Memory overflow--too many devices and/or file specifications (usually *.* operations) and no room for buffers.          |
| ?DEV FUL?     | No room on device for file.   |
| ?ER RD DIR?   | Unrecoverable error reading directory. Check volume for off-line or write-locked condition and try the operation again. |
| ?ER WR DIR?   | Unrecoverable error writing directory. Try again.   |
| ?EXT NEG?     | A /T command attempted to make file smaller.  |
| ?FG PRESENT?  | An attempt was made to use /O or /S while a foreground job was still in memory. Unload it if it is no longer desired.   |
| ?FIL NOT FND? | File not found during a delete, copy, or rename operation.  |



## Peripheral Interchange Program

|           |   |
|-----------|---|
| ?ILL CMD? | The command specified was not syntactically correct; a device name is missing which should be specified, a switch argument is too large, a filename is specified where one is inappropriate, or a non-file structured device is specified for a file-structured operation.  |
| ?ILL DEV? | Illegal or nonexistent device.  |
| ?ILL SWT? | Illegal switch or switch combination.   |
| ?ILL REN? | Illegal rename operation. Usually caused by different device names on the input and output sides of the command string.   |
| ?IN ER?   | Unrecoverable error reading file. Try again (this error is ignored during /G operation).  |
| ?OUT ER?  | Unrecoverable error writing file. Perhaps a hardware or checksum error; try recopying file. Also may be caused by an attempt to compress a larger device to a smaller one or by not enough room when creating a file. The system takes the largest space available and divides it in half before attempting to insert the file. Try the [] construction or /X switch. |
| ?OUT FIL? | Illegal output file specification or missing output file.   |
| ?ROOM?    | Insufficient space following file specified with a /T switch.   |

The following warning messages are output by PIP:

CTn: PUSH REWIND OR MOUNT NEW VOLUME

A new cassette must be mounted on drive n to allow continuation of an I/O operation. The operation is continued automatically as soon as the new cassette is mounted.

?NO SYS ACTION?

The /Y switch was not included with a command specified on a .SYS file. The command is executed for all but the .SYS files. A \*.\* transfer is most likely to cause this message.

?REBOOT?

.SYS files have been transferred, renamed, compressed or deleted from the system device. It may be necessary to reboot the system.

### NOTE

The message is typed immediately after execution of the relevant command has begun, but the actual reboot operation must not



## Peripheral Interchange Program

be performed until PIP returns with the prompting asterisk for the next command. If the system is halted and rebooted before the prompting asterisk returns, disk information may be lost.

If any of the .SYS files in use by the current system (MONITR.SYS and handler files) have been physically moved on the system device, it is necessary to reboot the system immediately. If not, this message can be ignored. If the cause of the message was a /S operation, the system need be rebooted only if there was an empty space before any of the .SYS files or if the /N:n switch was used to increase the number of directory segments. The need to reboot can be permanently avoided by placing all .SYS files at the beginning of the system device, then avoiding their involvements in PIP operations by not using the /Y switch.



## CHAPTER 5

### MACRO ASSEMBLER

MACRO is a 2-pass macro assembler requiring an RT-11 system configuration (or background partition) of 12K or more. Macros are instructions in a source or command language which are equivalent to a specified sequence of machine instructions or commands. Users with minimum memory configurations must use ASEMBL and EXPAND and should read this chapter and Chapters 10 and 11 before assembling any programs. (The macro features not supported by ASEMBL are indicated in this chapter; many of the features not available in ASEMBL are supported by EXPAND.)

Some notable features of MACRO are:

1. Program control of assembly functions
2. Device and file name specifications for input and output files
3. Error listing on command output device
4. Alphabetized, formatted symbol table listing
5. Relocatable object modules
6. Global symbols declaration for linking among object modules
7. Conditional assembly directives
8. Program sectioning directives
9. User defined macros
10. Comprehensive set of system macros
11. Extensive listing control, including cross reference listing

Operating instructions for the MACRO assembler appear in Section 5.7.



## MACRO Assembler

### 5.1 SOURCE PROGRAM FORMAT

A source program is composed of a sequence of source lines; each source line contains a single assembly language statement followed by a statement terminator. A terminator may be either a line feed character (which increments the line count by 1) or a form feed character (which increments both the line count and page count by 1).

#### NOTE

EDIT automatically appends a line feed to every carriage return encountered in a source program. For listing format, MACRO automatically inserts a carriage return before any line feed or form feed not already preceded by one.

An assembly language line can contain up to 132(decimal) characters (exclusive of the statement terminator). Beyond this limit, excess characters are ignored and generate an error flag.

#### 5.1.1 Statement Format

A statement can contain up to four fields which are identified by order of appearance and by specified terminating characters. The general format of a MACRO assembly language statement is:

```
label:    operator operand(s) ;comments
```

The label and comment fields are optional. The operator and operand fields are interdependent; either may be omitted depending upon the contents of the other.

The assembler interprets and processes these statements one by one, generating one or more binary instructions or data words or performing an assembly process. A statement contains one of these fields and may contain all four types. Blank lines are legal.

Some statements have one operand, for example:

```
CLR      R0
```

while others have two:

```
MOV      #344,R2
```

An assembly language statement must be complete on one source line. No continuation lines are allowed. (If a continuation is attempted with a line feed, the assembler interprets this as the statement terminator.)

MACRO source statements may be formatted with EDIT so that use of the TAB character causes the statement fields to be aligned. For example:



## MACRO Assembler

| <u>Label<br/>Field</u> | <u>Operator<br/>Field</u> | <u>Operand<br/>Field</u> | <u>Comment<br/>Field</u> |
|------------------------|---------------------------|--------------------------|--------------------------|
| CHECK:                 | BIT                       | #1,R0                    | ;IS NUMBER ODD?          |
|                        | BEQ                       | EVEN                     | ;NO, IT'S EVEN           |
|                        | MOV                       | #-1,ODDFLG               | ;ELSE SET FLAG           |
| EVEN:                  | RTS                       | PC                       | ;RETURN                  |

5.1.1.1 Label Field - A label is a user-defined symbol that is unique within the first six characters and is assigned the value of the current location counter and entered into the user-defined symbol table. The value of the label may be either absolute (fixed in memory independently of the position of the program) or relocatable (not fixed in memory), depending on whether the location counter value (see Section 5.2.6) is currently absolute or relocatable.

A label is a symbolic means of referring to a specific location within a program. If present, a label always occurs first in a statement and must be terminated by a colon. For example, if the current location is absolute 100(octal), the statement:

```
ABCD:  MOV    A,B
```

assigns the value 100(octal) to the label ABCD. Subsequent reference to ABCD references location 100(octal). In this example if the location counter was declared relocatable within the section, the final value of ABCD would be 100(octal) plus a value assigned by LINK when it relocates the code, called the relocation constant. (The final value of ABCD would therefore not be known until link-time. This is discussed later in this chapter and in Chapter 6.)

More than one label may appear within a single label field, in which case each label within the field is assigned the same value. For example, if the current location counter is 100(octal), the multiple labels in the statement:

```
ABC:   ERREX: MASK:  MOV    A,B
```

cause each of the three labels--ABC, ERREX, and MASK--to be equated to the value 100(octal).

A symbol used as a label may not be redefined within the user program. An attempt to redefine a label results in an error flag in the assembly listing.

5.1.1.2 Operator Field - An operator field follows the label field in a statement and may contain a macro call, an instruction mnemonic, or an assembler directive. The operator may be preceded by zero, one or more labels and may be followed by one or more operands and/or a comment. Leading and trailing spaces and tabs are ignored.

When the operator is a macro call, the assembler inserts the appropriate code to expand the macro. When the operator is an instruction mnemonic, it specifies the instruction to be generated and the action to be performed on any operand(s) which follow. When the operator is an assembler directive, it specifies a certain function or action to be performed during assembly.



## MACRO Assembler

An operator is legally terminated by a space, tab, or any non-alphanumeric character (symbol component).

Consider the following examples:

```
MOV A,B    (space terminates the operator MOV)
MOV@A,B    (@ terminates the operator MOV)
```

When the statement line does not contain an operand or comment, the operator is terminated by a carriage return followed by a line feed or form feed character.

A blank operator field is interpreted as a .WORD assembler directive (See Section 5.5.3.2).

**5.1.1.3 Operand Field** - An operand is that part of a statement which is manipulated by the operator. Operands may be expressions, numbers, or symbolic or macro arguments (within the context of the operation). When multiple operands appear within a statement, each is separated from the next by one of the following characters: comma, tab, space, or paired angle brackets around one or more operands (see Section 5.2.1.1). Multiple delimiters separating operands are not legal (with the exception of spaces and tabs--any combination of spaces and/or tabs represents a single delimiter). An operand may be preceded by an operator, a label or another operand and followed by a comment.

The operand field is terminated by a semicolon when followed by a comment, or by a statement terminator when the operand completes the statement. For example:

```
LABEL: MOV A,B ;COMMENT
```

The space between MOV and A terminates the operator field and begins the operand field; a comma separates the operands A and B; a semicolon terminates the operand field and begins the comment field.

**5.1.1.4 Comment Field** - The comment field is optional and may contain any ASCII characters except null, rubout, carriage return, line feed, vertical tab or form feed. All other characters, even special characters with defined usage, are ignored by the assembler when appearing in the comment field.

The comment field may be preceded by one, any, none or all of the other three field types. Comments must begin with the semicolon character and end with a statement terminator.



## MACRO Assembler

Comments do not affect assembly processing or program execution, but are useful in source listings for later analysis, debugging, or documentation purposes.

### 5.1.2 Format Control

Horizontal or line formatting of the source program is controlled by the space and tab characters. These characters have no effect on the assembly process unless they are embedded within a symbol, number, or ASCII text; or unless they are used as the operator field terminator. Thus, these characters can be used to provide an orderly source program. A statement can be written:

```
LABEL MOV(SP)+,TAG,POP VALUE OFF STACK
```

or, using formatting characters, it can be written:

```
LABEL: MOV      (SP)+,TAG      IPOP VALUE OFF STACK
```

which is easier to read in the context of a source program listing.

Vertical formatting, i.e., page size, is controlled by the form feed character. A page of n lines is created by inserting a form feed (CTRL FORM) after the nth line. (See also Section 5.5.1.6 for a description of page formatting with respect to macros and Section 5.5.1.2 for a description of assembly listing output.)

## 5.2 SYMBOLS AND EXPRESSIONS

This section describes the various components of legal MACRO expressions: the assembler character set, symbol construction, numbers, operators, terms and expressions.

### 5.2.1 Character Set

The following characters are legal in MACRO source programs:

1. The letters A through Z. Both upper- and lower-case letters are acceptable, although, upon input, lower-case letters are converted to upper-case letters. Lower-case letters can only be output by sending their ASCII values to the output device. This conversion is not true for .ASCII, .ASCIIZ, ' (single quote) or " (double quote) statements if .ENABL LC is in effect.
2. The digits 0 through 9.
3. The characters . (period or dot) and \$ (dollar sign) which are reserved for use in system program symbols (with the exception of local symbols; see Section 5.2.5).
4. The following special characters:



# MACRO Assembler

| <u>Character</u> | <u>Designation</u>  | <u>Function</u>   |
|------------------|---------------------|---|
| carriage return  |                     | formatting character  |
| line feed        |                     |   |
| form feed        |                     | source statement terminators                                |
| vertical tab     |                     |   |
| :                | colon               | label terminator  |
| =                | equal sign          | direct assignment indicator                                 |
| %                | percent sign        | register term indicator                                     |
| tab              |                     | item or field terminator                                    |
| space            |                     | item or field terminator                                    |
| #                | number sign         | immediate expression indicator                              |
| @                | at sign             | deferred addressing indicator                               |
| (                | left parenthesis    | initial register indicator                                  |
| )                | right parenthesis   | terminal register indicator                                 |
| ,                | comma               | operand field separator                                     |
| ;                | semicolon           | comment field indicator                                     |
| <                | left angle bracket  | initial argument or expression indicator                    |
| >                | right angle bracket | terminal argument or expression indicator                   |
| +                | plus sign           | arithmetic addition operator or auto increment indicator    |
| -                | minus sign          | arithmetic subtraction operator or auto decrement indicator |
| *                | asterisk            | arithmetic multiplication operator                          |
| /                | slash               | arithmetic division operator                                |
| &                | ampersand           | logical AND operator  |
| !                | exclamation         | logical inclusive OR operator                               |
| "                | double quote        | double ASCII character indicator                            |
| '                | single quote        | single ASCII character indicator                            |
| ↑                | uparrow             | universal unary operator, argument indicator                |
| \                | backslash           | macro numeric argument indicator (not available in ASEMBL)  |

5.2.1.1 Separating and Delimiting Characters - Reference is made in the remainder of the chapter to legal separating characters and macro argument delimiters. These terms are defined in Table 5-1 and following.

Table 5-1  
Legal Separating Characters

| Character | Definition                     | Usage   |
|-----------|--------------------------------|---|
| space     | one or more spaces and/or tabs | A space is a legal separator only for argument operands. Spaces within expressions are ignored. |
| ,         | comma                          | A comma is a legal separator for both expressions and argument operands.                        |
| <...>     | paired angle brackets          | Paired angle brackets are used to enclose an argument,  |

(Continued on next page)



Table 5-1 (cont.)  
Legal Separating Characters

| Character           | Definition   | Usage   |
|---------------------|--|---|
| <code>↑\...\</code> | Up arrow construction where the up arrow character is followed by an argument bracketed by any paired printing characters. | <p>particularly when that argument contains separating characters. Paired angle brackets may be used anywhere in a program to enclose an expression for treatment as a term.</p> <p>This construction is equivalent in function to the paired angle brackets and is generally used only where the argument contains angle brackets.</p> |

Macro arguments may appear in several forms to allow for special cases. The rules to observe when separating arguments are:

1. If an argument string contains only non-separating characters (those not defined in Table 5-1) and no spaces, then it may appear in the argument list separated, if necessary, from the other arguments by commas.
2. If an argument string contains separating characters or spaces, but does not contain the characters < or > (left or right angle brackets), then the argument may appear enclosed in paired angle brackets (e.g., <argument string>). The paired angle brackets are removed before the argument string is used. Successive pairs of angle brackets may be used to enclose an argument; only the outermost pair is removed.
3. If an argument string contains separating characters or spaces (possibly including the left or right angle bracket characters), then it may appear in the following form: `↑\argument string\` where the backslashes may be replaced by any character not appearing in the argument string. The uparrow and backslashes (or other character) are removed before the argument string is substituted into the text.

Note that regardless of the method used to specify an argument, it must be separated from any other arguments by commas.

#### 5.2.1.2 Illegal Characters - A character can be illegal in one of two ways:

1. A character which is not recognized as an element of the MACRO character set is always an illegal character and causes immediate termination of the current line at that point, plus the output of an error flag in the assembly listing. For example:

```
LABEL←*A: MOV A,B
```

Since the backarrow is not a recognized character, the entire line is treated as a:

```
.WORD LABEL
```

statement and is flagged in the listing.



## MACRO Assembler

2. A legal MACRO character may be illegal in context. Such a character generates a Q error on the assembly listing.

5.2.1.3 Operator Characters - Under MACRO, legal unary operators (operators applying to only one operand) are as follows:

| <u>Unary Operator</u> | <u>Explanation</u>   |            | <u>Example</u>  |
|-----------------------|--|------------|---|
| +                     | plus sign  | +A         | (positive value of A, equivalent to A)                                  |
| -                     | minus sign   | -A         | (negative, 2's complement, value of A)                                  |
| ↑                     | uparrow, universal unary operator (this usage is described in greater detail in Sections 5.5.4.2 and 5.5.6.2). | ↑F3.0      | (interprets 3.0 as a 1-word floating-point number)                      |
|                       |  | ↑C24       | (interprets the one's complement of the binary representation of 24(8)) |
|                       |  | ↑D127      | (interprets 127 as a decimal number)                                    |
|                       |  | ↑O34       | (interprets 34 as an octal number)                                      |
|                       |  | ↑B11000111 | (interprets 11000111 as a binary value)                                 |

The unary operators described above can be used adjacent to each other in a term. For example:

```
↑C↑O12
-↑O5
```

Legal binary operators under MACRO are as follows:

| <u>Binary Operator</u> | <u>Explanation</u>   | <u>Example</u>                 |
|------------------------|----------------------|--------------------------------|
| +                      | addition             | A+B                            |
| -                      | subtraction          | A-B                            |
| *                      | multiplication       | A*B (16-bit product returned)  |
| /                      | division             | A/B (16-bit quotient returned) |
| &                      | logical AND          | A&B                            |
| !                      | logical inclusive OR | A!B                            |

All binary operators have the same priority. Division and multiplication are signed operations. Items can be grouped for evaluation within an expression by enclosure in angle brackets. Terms in angle brackets are evaluated first, and remaining operations are performed left to right. For example:

```
.WORD 1+2*3 ;IS 11 OCTAL
.WORD 1+<2*3> ;IS 7 OCTAL
```



## MACRO Assembler

### 5.2.2 Symbols

There are three types of symbols: permanent, user-defined and macro. MACRO maintains three types of symbol tables: the Permanent Symbol Table (PST), the User Symbol Table (UST) and the Macro Symbol Table (MST). The PST contains all the permanent symbols and is part of the MACRO Assembler load module. The UST and MST are constructed as the source program is assembled; user-defined symbols are added to the table as they are encountered.

**5.2.2.1 Permanent Symbols -** Permanent symbols consist of the instruction mnemonics (Appendix C) and assembler directives and macro directives (sections 5.5 and 5.6, Appendix C). These symbols are a permanent part of the assembler and need not be defined before being used in the source program.

**5.2.2.2 User-Defined and Macro Symbols -** User-defined symbols are those used as labels or defined by direct assignment (Section 5.2.3). These symbols are added to the User Symbol Table as they are encountered during the first pass of the assembly. Macro symbols are those symbols used as macro names in the operator field (Section 5.6.1). These symbols are added to the Macro Symbol Table as they are encountered during the assembly.

User-defined and macro symbols can be composed of alphanumeric characters, dollar signs, and periods only; any other character is illegal.

The \$ and . characters are reserved for system software symbols (for example, the system macro symbol .READ); it is recommended that \$ and . not be inserted in user-defined or macro symbols.

The following rules apply to the creation of user-defined and macro symbols:

1. The first character must not be a number (except in the case of local symbols, see Section 5.2.5).
2. Each symbol must be unique within the first six characters.
3. A symbol can be written with more than six legal characters, but the seventh and subsequent characters are only checked for legality, and are not otherwise recognized by the assembler.
4. Spaces, tabs, and illegal characters must not be embedded within a symbol.

The value of a symbol depends upon its use in the program. A symbol in the operator field may be any one of the three symbol types. To determine the value of the symbol, the assembler searches the three symbol tables in the following order:

1. Macro Symbol Table
2. Permanent Symbol Table
3. User-Defined Symbol Table



## MACRO Assembler

A symbol found in the operand field is sought in the:

1. User-Defined Symbol Table
2. Permanent Symbol Table

in that order. The assembler never expects to find a macro name in an operand field.

These search orders allow redefinition of Permanent Symbol Table entries as user-defined or macro symbols. The same name can be assigned to both a macro and a label.

User-defined symbols are either internal or external (global). All user-defined symbols are internal unless explicitly defined as being global with the .GLOBL directive (see Section 5.5.10).

Global symbols provide links between object modules. A global symbol is defined as a label is generally called an entry point (to a section of code). Such symbols are referenced from other object modules to transfer control throughout the load module (which may be composed of a number of object modules).

Since MACRO provides program sectioning capabilities (Section 5.5.9), two types of internal symbols must be considered:

1. Symbols that belong to the current program section, and
2. Symbols that belong to other program sections.

In both cases, the symbol must be defined within the current assembly; the significance of the distinction is critical in evaluating expressions involving type (2) above (see Section 5.2.9.)

### 5.2.3 Direct Assignment

A direct assignment statement associates a symbol with a value. When a direct assignment statement defines a symbol for the first time, that symbol is entered into the user symbol table and the specified value is associated with it. A symbol may be redefined by assigning a new value to a previously defined symbol. The latest assigned value replaces any previous value assigned to a symbol.

The general format for a direct assignment statement is:

symbol = expression

Symbols take on the relocatable or absolute attribute of their defining expression. However, if the defining expression is global, the symbol is not global unless explicitly defined as such in a .GLOBL directive. For example:

```
A=1           ;THE SYMBOL A IS EQUATED TO THE
              ;VALUE 1
```

```
B='A-1&MASKLOW ;THE SYMBOL B IS EQUATED TO THE
              ;VALUE OF THE EXPRESSION
```

```
C1          D=3           ;THE SYMBOL D IS EQUATED TO 3
```



## MACRO Assembler

```
E:      MOV      #1,ABLE ;LABELS C AND E ARE EQUATED TO THE  
                          ;LOCATION OF THE MOV COMMAND
```

The following conventions apply to direct assignment statements:

1. An equal sign (=) must separate the symbol from the expression defining the symbol value.
2. A direct assignment statement is usually placed in the operator field and may be preceded by a label and followed by a comment.

### NOTE

If the program jumps to or references the label of a direct assignment statement, it is actually referencing the following instruction statement. For example:

```
          .5,+1000  
C:      D=3  
E:      MOV #D,ABLE  
  
          :  
          :  
          JMP C
```

This code causes a jump to the label E.

3. Only one symbol can be defined by any one direct assignment statement.
4. Only one level of forward referencing is allowed. That is, the following arrangement is illegal:

```
X = Y  
Y = Z  
Z = 1
```

X and Y are both undefined throughout pass 1. X is undefined throughout pass 2 and causes an error flag in the assembly listing.

### 5.2.4 Register Symbols

The eight general registers of the PDP-11 are numbered 0 through 7 and can be expressed in the source program as:

```
%0  
%1  
:  
:  
:  
%7
```



## MACRO Assembler

The digit indicating the specific register can be replaced by any legal term which can be evaluated during the first assembly pass.

It is recommended that the programmer create and use symbolic names for all register references. A register symbol may be defined in a direct assignment statement among the first statements in the program. A register symbol cannot be defined after the statement which uses it. The defining expression of a register symbol must be absolute. For example:

|       | REGISTER DEFINITION |
|-------|---------------------|
| R0=X0 |                     |
| R1=X1 |                     |
| R2=X2 |                     |
| R3=X3 |                     |
| R4=X4 |                     |
| R5=X5 |                     |
| SP=X6 |                     |
| PC=X7 |                     |

The symbolic names assigned to the registers in the example above are the conventional names used in all PDP-11 system programs. Since these names are fairly mnemonic, it is suggested the user follow this convention. Registers 6 and 7 are given special names because of their special functions, while registers 0 through 5 are given similar names to denote their status as general purpose registers.

All register symbols must be defined before they are referenced. A forward reference to a register symbol causes phase errors in an assembly.

The % character can be used with any term or expression to specify a register. (A register expression less than 0 or greater than 7 is flagged with an R error code.) For example:

```
CLR X3+1
```

is equivalent to:

```
CLR X4
```

and clears the contents of register 4, while:

```
CLR 4
```

clears the contents of memory address 4.

In certain cases a register can be referenced without the use of a register symbol or register expression; these cases are recognized through the context of the statement. An example is shown below:

|            |                                  |
|------------|----------------------------------|
| JSR 5,SUBR | IFIRST OPERAND FIELD MUST ALWAYS |
|            | BE A REGISTER                    |

### 5.2.5 Local Symbols

Local symbols are specially formatted symbols used as labels within a given range.



## MACRO Assembler

Local symbols provide a convenient means of generating labels to be referenced by branch instructions. Use of local symbols reduces the possibility of multiply-defined symbols within a user program and separates entry point symbols from local references. Local symbols, then, are not referenced from other object modules or even from outside their local symbol block.

Local symbols are of the form  $n\$$ , where  $n$  is a decimal integer from 1 to 127, inclusive, and can only be used on word boundaries. Local symbols include:

1\$  
27\$  
59\$  
104\$

Within a local symbol block, local symbols can be defined and referenced. However, a local symbol cannot be referenced outside the block in which it is defined. There is no conflict with labels of the same name in other local symbol blocks.

Local symbols 64\$ through 127\$ can be generated automatically as a feature of the macro processor (see Section 5.6.3.5 for further details). When using local symbols the user is advised to first use the range from 1\$ to 63\$.

A local symbol block is delimited in one of the following ways:

1. The range of a single local symbol block can consist of those statements between two normally constructed symbolic labels. (Note that a statement of the form:

LABEL=.

is a direct assignment, does not create a label in the strict sense, and does not delimit a local range.)

2. The range of a local symbol block is terminated upon encountering a .CSECT directive.
3. The range of a single local symbol block can be delimited with .ENABL LSB and the first symbolic label or .CSECT directive following the .DSABL LSB directives. The default for LSB is off.

For examples of local symbols and local symbol blocks, see Figure 5-1.

The maximum offset of a local symbol from the base of its local symbol block is 128 decimal words. Symbols beyond this range are flagged with an A error code.



## MACRO Assembler

### 5.2.6 Assembly Location Counter

The period (.) is the symbol for the assembly location counter. When used in the operand field of an instruction, it represents the address of the first word of the instruction. When used in the operand field of an assembler directive, it represents the address of the current byte or word. For example:

```

      A:      MOV      #.,R0      ;. REFERS TO LOCATION A,
                                   ;I.E., THE ADDRESS OF THE
                                   ;MOV INSTRUCTION
```

(# is explained in Section 5.4.9).

At the beginning of each assembly pass, the assembler clears the location counter. Normally, consecutive memory locations are assigned to each byte of object data generated. However, the location where the object data is stored may be changed by a direct assignment statement altering the location counter:

```
      .=expression
```

The expression defining the location counter must not contain forward references or symbols that vary from one pass to another. If an expression is assigned to the current location counter in a relocatable CSECT, an error flag is generated. (The construction `.=.+expression` must be used.)

Similar to other symbols, the location counter symbol has a mode associated with it, either absolute or relocatable; the mode cannot be external. The existing mode of the location counter cannot be changed by using a defining expression of a different mode.



# MACRO Assembler

| Line<br>Number | Octal<br>Expansion | Source Code           | Comments              |
|----------------|--------------------|-----------------------|-----------------------|
| 1              |                    |                       |                       |
| 2              |                    |                       |                       |
| 3              |                    |                       |                       |
| 4              |                    |                       |                       |
| 5              |                    |                       |                       |
| 6              |                    | .MCALL .REGDEF,..V2.. |                       |
| 7 000000       |                    | .REGDEF               |                       |
| 8 000000       |                    | ..V2..                |                       |
| 9              | 000000             | R0=X0                 |                       |
| 10             |                    | .SBTTL                | SECTOR INITIALIZATION |
| 11             | 000000'            | .CSECT                | IMPURE                |
| 12 000000      | IMPURE:            |                       | IMPURE STORAGE AREA   |
| 13             | 000000'            | .CSECT                | IMPPAS                |
| 14 000000      | IMPPAS:            |                       | ICLEARED EACH PASS    |
| 15             | 000000'            | .CSECT                | IMPLIN                |
| 16 000000      | IMPLIN:            |                       | ICLEARED EACH LINE    |
| 17             | 000000'            | .CSECT                | XCTPRG                |
| 18             |                    |                       | PROGRAM               |
| 19 000000      | XCTPRG:            |                       | INITIALIZATION        |
| 20 000000      | 012700             | MOV                   | #IMPURE,R0            |
|                | 000000'            |                       |                       |
| 21 000004      | 005020 1S:         | CLR                   | (R0)+                 |
| 22 000006      | 022700             | CMP                   | #IMPTOP,R0            |
|                | 000000'            |                       | ICLEAR IMPURE AREA    |
| 23 00012       | 101374             | BHI                   | 1S                    |
| 24             |                    |                       |                       |
| 25             | 000000'            | .CSECT                | XCTPAS                |
| 26 000000      | XCTPAS:            |                       | PASS INITIALIZATION   |
| 27 000000      | 012700             | MOV                   | #IMPPAS,R0            |
|                | 000000'            |                       |                       |
| 28 000004      | 005020 1S:         | CLR                   | (R0)+                 |
| 29 000006      | 022700             | CMP                   | #IMPTOP,R0            |
|                | 000000'            |                       | ICLEAR IMPURE PART    |
| 30 00012       | 101374             | BHI                   | 1S                    |
| 31             |                    |                       |                       |
| 32             | 000000'            | .CSECT                | XCTLIN                |
| 33 000000      | XCTLIN:            |                       | LINE INITIALIZATION   |
| 34 000000      | 012700             | MOV                   | #IMPLIN,R0            |
|                | 000000'            |                       |                       |
| 35 000004      | 005020 1S:         | CLR                   | (R0)+                 |
| 36 000006      | 022700             | CMP                   | #IMPTOP,R0            |
|                | 000000'            |                       |                       |
| 37 00012       | 101374             | BHI                   | 1S                    |
| 38             |                    |                       |                       |
| 39             | 000000'            | .CSECT                | MIXED                 |
| 40 000000      | 000000 IMPTOP:     | .WORD 0               | MIXED MODE SECTOR     |
| 41             | 000001'            | .END                  |                       |

Figure 5-1  
Assembly Source Listing of MACRO Code Showing Local Symbol Blocks



## MACRO Assembler

The mode of the location counter symbol can be changed by the use of the .ASECT or .CSECT directive as explained in Section 5.5.9.

Examples:

```
.ASECT
    =500                                ;SET LOCATION COUNTER TO
                                        ;ABSOLUTE 500

FIRST: MOV ,+10,COUNT                  ;THE LABEL FIRST HAS THE VALUE
                                        ;500(8)
                                        ;+,+10 EQUALS 510(8). THE
                                        ;CONTENTS OF THE LOCATION
                                        ;510(8) WILL BE DEPOSITED
                                        ;IN LOCATION COUNT.

COUNT: .WORD 0

    =520                                ;THE ASSEMBLY LOCATION COUNTER
                                        ;NOW HAS A VALUE OF
                                        ;ABSOLUTE 520(8).

SECOND: MOV ,,INDEX                   ;THE LABEL SECOND HAS THE
                                        ;VALUE 520(8)
                                        ;THE CONTENTS OF LOCATION
                                        ;520(8), THAT IS, THE BINARY
                                        ;CODE FOR THE INSTRUCTION
                                        ;ITSELF WILL BE DEPOSITED IN
                                        ;LOCATION INDEX.

INDEX: .WORD 0

.CSECT
    =,+20                              ;SET LOCATION COUNTER TO
                                        ;RELOCATABLE 20 OF THE
                                        ;UNNAMED PROGRAM SECTION.

THIRD: .WORD 0                        ;THE LABEL THIRD HAS THE
                                        ;VALUE OF RELOCATABLE 20.
```

Storage area may be reserved by advancing the location counter. For example, if the current value of the location counter is 1000, the direct assignment statement:

```
    =,+100
```

reserves 100(octal) bytes of storage space in the program. The next instruction is stored at 1100. (The .BLKW and .BLKB directives can also be used to reserve blocks of storage; see Section 5.5.5.3.)



## MACRO Assembler

### 5.2.7 Numbers

The MACRO Assembler assumes all numbers in the source program are to be interpreted in octal radix unless otherwise specified. The assumed radix can be altered with the .RADIX directive or individual numbers can be treated as being of decimal, binary, or octal radix (see Section 5.5.4.2).

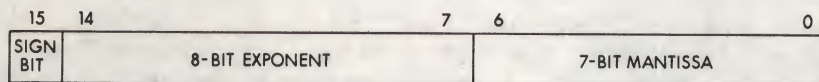
Octal numbers consist of the digits 0 through 7 only. A number not specified as a decimal number and containing an 8 or 9 is flagged with an N error code and treated as a decimal number.

Negative numbers are preceded by a minus sign (the assembler translates them into two's complement form). Positive numbers may be preceded by a plus sign, although this is not required.

A number which is too large to fit into 16 bits (177777<n) is truncated from the left and flagged with a T error code in the assembly listing.

Numbers are always considered absolute quantities (that is, not relocatable).

The single-word floating-point numbers which can be generated with the  $\uparrow$ F operator (see Section 5.5.6.2) are stored in the following format:



Refer to PDP-11/45 PROCESSOR HANDBOOK for details of the floating-point format.

### 5.2.8 Terms

A term is a component of an expression. A term may be one of the following:

1. A number whose 16-bit value is used.
2. A symbol that is interpreted according to the following hierarchy:
  - a. a period that causes the value of the current location counter to be used
  - b. a permanent symbol whose basic value is used and whose arguments (if any) are ignored
  - c. user defined symbols
  - d. an undefined symbol that is assigned a value of zero and inserted in the user-defined symbol table



## MACRO Assembler

3. An ASCII conversion using either an apostrophe followed by a single ASCII character or a double quote followed by two ASCII characters, which results in a word containing the 7-bit ASCII value of the character(s). (This construction is explained in greater detail in Section 5.5.3.3.)
4. An expression enclosed in angle brackets. Any quantity enclosed in angle brackets is evaluated before the remainder of the expression in which it is found. Angle brackets are used to alter the left to right evaluation of expressions (for example, to differentiate between  $A*B+C$  and  $A*(B+C)$ ) or to apply a unary operator to an entire expression ( $-(A+B)$ ).

### 5.2.9 Expressions

Expressions are combinations of terms that are joined together by binary operators and that reduce to a 16-bit value. The operands of a .BYTE directive are evaluated as word expressions before truncation to the low-order eight bits. Prior to truncation, the high-order byte must be zero or all ones (when the byte value is negative, the sign bit is propagated). The evaluation of an expression includes the evaluation of the mode of the resultant expression--that is, absolute, relocatable or external. Expression modes are defined further below.

Expressions are evaluated left to right with no operator hierarchy rules except that unary operators take precedence over binary operators. A term preceded by a unary operator can be considered as containing that unary operator. (Terms are evaluated, where necessary, before their use in expressions.) Multiple unary operators are valid and are treated as follows:

--+A

is equivalent to:

-(<+(-A)>)

The value of an external expression is the value of the absolute part of the expression; e.g.,  $EXT+A$  has a value of A. This is modified by the Linker to become  $EXT+A$ .

Expressions, when evaluated, are either absolute, relocatable, or external. For the programmer writing position independent code, the distinction is important.

1. An expression is absolute if its value is fixed. An expression whose terms are numbers and ASCII conversions has an absolute value. A relocatable expression minus a relocatable term, where both items belong to the same program section, is also absolute.
2. An expression is relocatable if its value is fixed relative to a base address but will have an offset value added when linked. Expressions whose terms contain labels defined in relocatable sections and the assembly location counter (in relocatable sections) have a relocatable value.
3. An expression is external (or global) if its value is only partially defined during assembly and is completed at link



## MACRO Assembler

time. An expression whose terms contain a global symbol not defined in the current program is an external expression. External expressions have relocatable values at execution time if the global symbol is defined as being relocatable or absolute if the global symbol is defined as absolute.

An example of the three expression types follows:

```
.ASECT
    =100
ABSSYM=.                                ;THE VALUE OF ABSSYM IS
                                        ;NOT RELOCATABLE, BECAUSE
                                        ;WE ARE IN AN ASECT

    .CSECT MAIN                          ;START RELOCATABLE
                                        ;PROGRAM SECTION

    .GLOBL EXTVAL                       ;EXTVAL IS DEFINED ELSEWHERE,
                                        ;ITS VALUE WILL NOT BE KNOWN
                                        ;UNTIL LINK TIME

BEGSYM: .BLKW 4                          ;THE VALUES OF BEGSYM
    .ASCII /ABCD/                       ;AND ENDSYM ARE
    .EVEN                               ;RELOCATABLE, BECAUSE
ENDSYM=.                                ;THE ADDRESS AT WHICH
                                        ;"MAIN" WILL BE LOADED
                                        ;IS NOT DETERMINED UNTIL
                                        ;LINK TIME

SIZE = ENDSYM-BEGSYM                    ;HOWEVER, THE
                                        ;VALUE OF SIZE IS KNOWN
                                        ;(IT IS 12.)AT ASSEMBLY
                                        ;TIME AND IS ABSOLUTE

REEXP = ENDSYM-BEGSYM+.                  ;REEXP (=,+12.) IS
                                        ;RELOCATABLE

EXTEXP: .WORD EXTVAL+4                  ;THE EXPRESSION "EXTVAL+4"
                                        ;IS EXTERNAL (OR GLOBAL)
                                        ;BECAUSE EXTVAL IS DEFINED
                                        ;IN ANOTHER PROGRAM UNIT.

CHARA='A'                               ;THE VALUE OF CHARA
                                        ;IS ABSOLUTE
```

### 5.3 RELOCATION AND LINKING

The output of the MACRO Assembler is an object module which must be processed by LINK before loading and execution (refer to Chapter 6 for details). The Linker essentially fixes (i.e., makes absolute) the values of external or relocatable symbols and turns the object module into a load module.

To enable the Linker to fix the value of an expression, the assembler issues certain directives to the Linker together with required parameters. In the case of relocatable expressions, the Linker adds the base of the associated relocatable section (the location in memory of relocatable 0) to the value of the relocatable expression provided



## MACRO Assembler

by the assembler. In the case of an external expression, the value of the external term in the expression is determined by the Linker (since the external symbol must be defined in one of the other object modules which are being linked together) and adds it to the value of the external expression provided by the assembler.

All words that are to be modified (as described in the previous paragraph) are marked with an apostrophe in the assembly listing. A G in the listing indicates that the value is external, or that a global is added to that value. Thus, the binary text output looks as follows:

|         |     |                 |                                |
|---------|-----|-----------------|--------------------------------|
| 005065  | CLR | EXTERNAL(R5)    | /VALUE OF EXTERNAL SYMBOL      |
| 000000G |     |                 | /ASSEMBLED ZERO/ WILL BE       |
|         |     |                 | /MODIFIED BY THE LINKER,       |
| 005065  | CLR | EXTERNAL+6(R5)  | /THE ABSOLUTE PORTION OF THE   |
| 000006G |     |                 | /EXPRESSION (000006) IS ADDED  |
|         |     |                 | /BY THE LINKER TO THE VALUE OF |
|         |     |                 | /THE EXTERNAL SYMBOL           |
| 005065  | CLR | RELOCATABLE(R5) | /ASSUMING WE ARE IN A          |
| 000040  |     |                 | /RELOCATABLE SECTION           |
|         |     |                 | /AND THE VALUE OF RELOCATABLE  |
|         |     |                 | /IS RELOCATABLE 40             |

### 5.4 ADDRESSING MODES

The program counter (PC, register 7 of the eight general registers) always contains the address of the next word to be fetched; i.e., the address of the next instruction to be executed, or the second or third word of the current instruction.

In order to understand how the address modes operate and how they assemble, the action of the program counter must be understood. The key rule is:

Whenever the processor implicitly uses the program counter to fetch a word from memory, the program counter is automatically incremented by two after the fetch.

That is, when an instruction is fetched, the PC is incremented by two so that it is pointing to the next word in memory; if an instruction uses indexing (Sections 5.4.7, 5.4.9 and 5.4.11) the processor uses the program counter to fetch the base from memory. Hence, using the rule above, the PC increments by two, and now points to the next word.

The following conventions are used in this section:

1. Let E be any expression as defined in Section 5.2.
2. Let R be a register expression. This is any expression containing a term preceded by a % character or a symbol previously equated to such a term.



## MACRO Assembler

### Examples:

```
R0=X0    ;GENERAL REGISTER 0
R1=R0+1  ;GENERAL REGISTER 1
R2=1+X1  ;GENERAL REGISTER 2
```

3. Let ER be a register expression or an expression in the range 0 to 7 inclusive.
4. Let A be any general address specification which produces a 6-bit mode address field as described in Sections 3.1 and 3.2 of the PDP-11 PROCESSOR HANDBOOK (both 11/20 and 11/45 versions).

The addressing specifications, A, can be explained in terms of E, R, and ER as defined above. Each is illustrated with the single operand instruction CLR or double operand instruction MOV.

#### 5.4.1 Register Mode

The register contains the operand.

Format for A: R

```
Examples:    R0=X0    ;DEFINE R0 AS REGISTER 0
              CLN      R0    ;CLEAR REGISTER 0
```

#### 5.4.2 Register Deferred Mode

The register contains the address of the operand.

Format for A: @R or (ER)

```
Examples:    CLN      @R1    ;BOTH INSTRUCTIONS CLEAR
              CLN      (R1)   ;THE WORD AT THE ADDRESS
                                ;CONTAINED IN REGISTER 1
```

#### 5.4.3 Autoincrement Mode

The contents of the register are incremented immediately after being used as the address of the operand. (See NOTE below.)



## MACRO Assembler

Format for A: (ER)+

Examples:

|      |         |                             |
|------|---------|-----------------------------|
| CLW  | (R0)+   | ;EACH INSTRUCTION CLEARS    |
| CLW  | (R0+3)+ | ;THE WORD AT THE ADDRESS    |
| CLW  | (R2)+   | ;CONTAINED IN THE SPECIFIED |
|      |         | ;REGISTER AND INCREMENTS    |
|      |         | ;THAT REGISTER'S CONTENTS   |
|      |         | ;BY TWO.                    |
| CLWB | (R4)+   | ;CLEARS THE BYTE AT THE     |
|      |         | ;ADDRESS SPECIFIED BY THE   |
|      |         | ;CONTENTS OF R4 AND         |
|      |         | ;INCREMENTS R4 BY ONE.      |

### NOTE

Both JMP and JSR instructions using non-deferred autoincrement mode, autoincrement the register before its use on the PDP-11/20 and 11/05 (but not on the PDP-11/40 or 11/45). In double operand instructions of the addressing form %R,(R)+ or %R,-(R) where the source and destination registers are the same, the source operand is evaluated as the autoincremented or autodecremented value, but the destination register, at the time it is used, still contains the originally intended effective address.

In the following two examples, as executed on the PDP-11/20, R0 originally contains 100.

|     |          |                            |
|-----|----------|----------------------------|
| MOV | R0,(R0)+ | ;THE QUANTITY 102 IS MOVED |
|     |          | ;TO LOCATION 100           |
| MOV | R0,=(R0) | ;THE QUANTITY 76 IS MOVED  |
|     |          | ;TO LOCATION 76            |

The use of these forms should be avoided as they are not compatible with the PDP-11/05, 11/40 and 11/45.

A Z error code is printed with each instruction which is not compatible among all members of the PDP-11 family. This is merely a warning code.

#### 5.4.4 Autoincrement Deferred Mode

The register contains the pointer to the address of the operand. The contents of the register are incremented after being used.



## MACRO Assembler

Format for A: @ (ER) +

Example:        CLR        @ (R3) +    ;CONTENTS OF REGISTER 3 POINT  
  ;TO ADDRESS OF WORD TO BE  
  ;CLEARED, AND REGISTER 3 IS  
  ;THEN INCREMENTED BY TWO

### 5.4.5 Autodecrement Mode

The contents of the register are decremented before being used as the address of the operand (see NOTE under autoincrement mode).

Format for A: - (ER)

Examples:        CLR        - (R0)    ;DECREMENT CONTENTS OF  
                  CLM        - (R0+3) ;0, 3, AND 2 BY TWO  
                  CLM        - (R2)    ;BEFORE USING AS ADDRESSES  
  ;OF WORDS TO BE CLEARED.

### 5.4.6 Autodecrement Deferred Mode

The contents of the register are decremented before being used as the pointer to the address of the operand.

Format for A: @ - (ER)

Example:        CLR        @ - (R2)    ;DECREMENT CONTENTS OF  
  ;REGISTER 2 BY TWO BEFORE  
  ;USING AS A POINTER  
  ;TO ADDRESS OF WORD TO BE  
  ;CLEARED.

### 5.4.7 Index Mode

The value of an expression E is stored as the second or third word of the instruction. The effective address is calculated as the value of E plus the contents of register ER. The value E is called the base.

Format for A: E (ER)

Examples:        CLR        X+2 (R1) ;EFFECTIVE ADDRESS IS X+2 PLUS  
  ;THE CONTENTS OF REGISTER 1  
                  CLR        -2 (R3) ;EFFECTIVE ADDRESS IS -2 PLUS  
  ;THE CONTENTS OF REGISTER 3,

### 5.4.8 Index Deferred Mode

An expression plus the contents of a register gives the pointer to the address of the operand.



## MACRO Assembler

Format for A: @E(ER)

Example:        CLR        #14(R4) ;IF REGISTER 4 HOLDS 100 AND  
                              ;LOC 114 HOLDS 2000,  
                              ;LOCATION 2000 IS CLEARED.

### 5.4.9 Immediate Mode

The immediate mode allows the operand itself to be stored as the second or third word of the instruction. It is assembled as an autoincrement of register 7, the PC.

Format for A: #E

Examples: MOV        #100,R0 ;MOVE AN OCTAL 100 TO  
                              ;REGISTER 0  
              MOV        #X,R0 ;MOVE THE VALUE OF THE SYMBOL X TO  
                              ;REGISTER 0

The operation of this mode can be explained by the following example. The statement MOV #100,R3 assembles as two words. These are:

012703  
000100

Just before this instruction is fetched and executed, the PC points to the first word of the instruction. The processor fetches the first word and increments the PC by two. The source operand mode is 27 (autoincrement the PC). Thus, the PC is used as a pointer to fetch the operand (the second word of the instruction) before being incremented by two, to point to the next instruction.

### 5.4.10 Absolute Mode

Absolute mode is the equivalent of immediate mode deferred. @#E specifies an absolute address which is stored in the second or third word of the instruction. Absolute mode is assembled as an autoincrement deferred of register 7, the PC.

Format for A: @#E

Examples: MOV        @#100,R0 ;MOVE THE VALUE OF CONTENTS  
                              ;OF LOCATION 100 TO  
                              ;REGISTER 0.  
              CLR        @#X    ;CLEAR THE CONTENTS OF THE  
                              ;LOCATION WHOSE ADDRESS IS X,

### 5.4.11 Relative Mode

Relative mode is the normal mode for memory references.



## MACRO Assembler

Format for A: E

Examples:    CLR        100        ;CLEAR LOCATION 100  
             MOV        X,Y        ;MOV THE CONTENTS OF LOCATION X  
                                 ;TO LOCATION Y.

Relative mode is assembled as index mode, using register 7, the PC, as the index register. The base of the address calculation, which is stored in the second or third word of the instruction, is not the address of the operand (as in index mode), but the number which, when added to the PC, becomes the address of the operand. Thus, the base is X-PC, which is called an offset. The operation is explained as follows:

If the statement MOV 100,R3 is assembled at absolute location 20, the assembled code is:

Location 20:        016703  
Location 22:        000054

The processor fetches the MOV instruction and adds two to the PC so that it points to location 22. The source operand mode is 67, that is, indexed by the PC. To pick up the base, the processor fetches the word pointed to by the PC and adds two to the PC. The PC now points to location 24. To calculate the address of the source operand, the base is added to the designated register, that is,  $\text{BASE} + \text{PC} = 54 + 24 = 100$ , the operand address.

Since the assembler considers "." as the address of the first word of the instruction, an equivalent index mode statement would be:

MOV 100-.-4(PC),R3

This mode is called relative because the operand address is calculated relative to the current PC. The base is the distance or offset (in bytes) between the operand and the current PC. If the operator and its operand are moved in memory so that the distance between the operator and data remains constant, the instruction will operate correctly anywhere in memory.

### 5.4.12 Relative Deferred Mode

Relative deferred mode is similar to relative mode, except that the expression, E, is used as the pointer to the address of the operand.

Format for A:        @E

Example:        MOV @X,R0        ;MOVE THE CONTENTS OF THE  
                                 ;LOCATION WHOSE ADDRESS IS IN  
                                 ;X INTO REGISTER 0

### 5.4.13 Table of Mode Forms and Codes

Each instruction assembles into at least one word. Operands of the first six forms listed below do not increase the length of an instruction. Each operand in one of the other modes, however, increases the instruction length by one word.



## MACRO Assembler

| <u>Form</u> | <u>Mode</u> | <u>Meaning</u>              |
|-------------|-------------|-----------------------------|
| R           | 0n          | Register mode               |
| @R or (ER)  | 1n          | Register deferred mode      |
| (ER)+       | 2n          | Autoincrement mode          |
| @(ER)+      | 3n          | Autoincrement deferred mode |
| -(ER)       | 4n          | Autodecrement mode          |
| @-(ER)      | 5n          | Autodecrement deferred mode |

n represents the register number.

Any of the following forms adds one word to the instruction length:

| <u>Form</u> | <u>Mode</u> | <u>Meaning</u>                   |
|-------------|-------------|----------------------------------|
| E (ER)      | 6n          | Index mode                       |
| @E(ER)      | 7n          | Index deferred mode              |
| #E          | 27          | Immediate mode                   |
| @#E         | 37          | Absolute memory reference mode   |
| E           | 67          | Relative mode                    |
| @E          | 77          | Relative deferred reference mode |

n represents the register number. Note that in the last four forms, register 7 (the PC) is referenced.

### NOTE

An alternate form for @R is (ER). However, the form @(ER) is equivalent to @0(ER).

The form @#E differs from the form E in that the second or third word of the instruction contains the absolute address of the operand rather than the relative distance between the operand and the PC. Thus, the instruction CLR @#100 clears absolute location 100 even if the instruction is moved from the point at which it was assembled. See the description of the .ENABL AMA function in Section 5.5.2, which directs the assembly of all relative mode addresses as absolute mode addresses.

#### 5.4.14 Branch Instruction Addressing

The branch instructions are 1-word instructions. The high byte contains the op code and the low byte contains an 8-bit signed offset which specifies the branch address relative to the PC. Upon execution of a branch instruction, the hardware calculates the branch address as follows:

1. Extend the sign of the offset through bits 8-15.
2. Multiply the result by 2. This creates a word offset rather than a byte offset.



## MACRO Assembler

3. Add the result to the PC to form the final branch address.

The assembler performs the reverse operation to form the byte offset from the specified address. Remember that when the offset is added to the PC, the PC is pointing to the word following the branch instruction; hence the term -2 in the calculation.

Byte offset =  $(E - PC) / 2$  truncated to eight bits.

Since PC = .+2, we have:

Byte offset =  $(E - .-2) / 2$  truncated to eight bits.

### NOTE

It is illegal to branch to a location specified as an external symbol, or to a relocatable symbol from within an absolute section, or to an absolute symbol or a relocatable symbol or another program section from within a relocatable section.

## 5.4.15 EMT and TRAP Addressing

The EMT and TRAP instructions do not use the low-order byte of the word. This allows information to be transferred to the trap handlers in the low-order byte. If EMT or TRAP is followed by an expression, the value is put into the low-order byte of the word. However, if the expression is too big (>377(8)) it is truncated to eight bits and a T error flag is generated.

## 5.5 ASSEMBLER DIRECTIVES

Directives are statements which cause the assembler to perform certain processing operations.

Assembler directives can be preceded by a label, subject to restrictions associated with specific directives, and followed by a comment. An assembler directive occupies the operator field of a MACRO source line. Only one directive can be placed on any one line. Zero, one, or more operands can occupy the operand field; legal operands differ with each directive and may be either symbols, expressions, or arguments.

### 5.5.1 Listing Control Directives

5.5.1.1 .LIST and .NLIST - Listing options can be specified in the text of a MACRO program through the .LIST and .NLIST directives. These are of the form:

.LIST arg  
.NLIST arg



## MACRO Assembler

where arg represents one or more optional arguments. When used without arguments, the listing directives alter the listing level count. The listing level count causes the listing to be suppressed when it is negative. The count is initialized to zero, incremented for each .LIST and decremented for each .NLIST. For example:

```
.LIST ME
.MACRO LTEST      ;LIST TEST
;A=THIS LINE SHOULD LIST
.NLIST
;B=THIS LINE SHOULD NOT LIST
.NLIST
;C=THIS LINE SHOULD NOT LIST
.LIST
;D=THIS LINE SHOULD NOT LIST (LEVEL NOT BACK TO ZERO)
.LIST
;E=THIS LINE SHOULD LIST (LEVEL BACK TO ZERO)
.ENDM
LTEST              ;CALL THE MACRO

;A=THIS LINE SHOULD LIST
;E=THIS LINE SHOULD LIST (LEVEL BACK TO ZERO)
```

The primary purpose of the level count is to allow macro expansions to be selectively listed and yet exit with the level returned to the status current during the macro call.

The use of arguments with the listing directives does not affect the level count; however, .LIST and .NLIST can be used to override the current listing control. For example:

```
.MACRO XX
:
:
.LIST      ;LIST NEXT LINE
X=.
.NLIST    ;DO NOT LIST REMAINDER
          ;OF MACRO EXPANSION
:
:
.ENDM
.NLIST ME      ;DO NOT LIST MACRO EXPANSIONS
XX
X=.
```

Allowable arguments for use with the listing directives are as follows (these arguments can be used singly or in combination):

| <u>Argument</u> | <u>Default</u> | <u>Function</u>   |
|-----------------|----------------|---|
| SEQ             | list           | Controls the listing of source line sequence numbers.                                       |
| LOC             | list           | Controls the listing of the location counter (this field would not normally be suppressed). |
| BIN             | list           | Controls the listing of generated binary code (supersedes BEX).                             |



## MACRO Assembler

|     |               |  |
|-----|---------------|--|
| BEX | list          | Controls listing of binary extensions; that is, prevents listing those locations and binary contents beyond the first line of an expansion. This is a subset of the BIN argument.  |
| SRC | list          | Controls the listing of the source code.   |
| COM | list          | Controls the listing of comments. This is a subset of the SRC argument and can be used to reduce listing time and/or space where comments are unnecessary.   |
| MD  | list          | Controls listing of macro definitions and repeat range expansions (has no effect in ASEMBL).   |
| MC  | list          | Controls listing of macro calls and repeat range expansions (has no effect in ASEMBL).   |
| ME  | no list       | Controls listing of macro expansions (supersedes MEB; has no effect in ASEMBL).  |
| MEB | no list       | Controls listing of macro expansion binary code. A .LIST MEB causes only those macro expansion statements producing binary code to be listed. This is a subset of the ME argument (has no effect in ASEMBL).   |
| CND | list          | Controls the listing of unsatisfied conditions and all .IF and .ENDC statements. This argument permits conditional assemblies to be listed without including unsatisfied code.   |
| LD  | no list       | Controls listing of all listing directives having no arguments (those used to alter the listing level count).  |
| TOC | list          | Controls listing of table of contents on pass 1 of the assembly (see Section 5.5.1.4 describing the .SBTTL directive). The full assembly listing is printed during pass 2 of the assembly.   |
| TTM | Teletype mode | Controls listing output format (has no effect in ASEMBL). The TTM argument (the default case) causes output lines to be truncated to 72 characters. Binary code is printed with the binary extensions below the first binary word. The alternative (.NLIST TTM) to Terminal mode is line printer mode, which is shown in Figure 5-2. |
| SYM | list          | Controls the listing of the symbol table for the assembly.   |



## MACRO Assembler

An example of an assembly listing as sent to a 132-column line printer is shown in Figure 5-2. Notice that binary extensions for statements generating more than one word are spread horizontally on the source line. An example of an assembly listing as sent to an 80-column line printer is shown in Figure 5-3 (this is the same format as a terminal listing). Notice that binary extensions for statements generating more than one word are printed on subsequent lines.

Figure 5-4 illustrates a symbol table listing. With the exception of local symbols and macro names, all user-defined symbols are listed in the symbol table. The characters following the symbols listed have special meanings as follows:

|   |   |
|---|---|
| = | the symbol is assigned in a direct assignment statement |
| % | the symbol is a register symbol                         |
| R | the symbol is relocatable                               |
| G | the symbol is global                                    |

The final value of the symbol is expressed in octal. If the symbol is undefined six asterisks are printed in place of the octal number.

CSECT numbers are listed if the symbol is in a named CSECT. All CSECTs are listed at the end of the table with their lengths and corresponding number.



Figure 5-2  
Example of MACRO Line Printer Listing  
(132-column Line Printer)



RTEXEC RT-11 MACRO VM02-09 5-SEP-74 22:30:23 PAGE 21  
GET PHYSICAL SOURCE LINE

```

1      .SBTTL GET PHYSICAL SOURCE LINE
2
3      104240      WINST=EMT+240
4      001756      GETPLI:
5      001756      SREADW SRC
6      001766      CLR R0
7      001770      BIT #10.EOF,IOFTBL+SRCCHN ;END OF FILE?
8      001776      BEQ 2$ ;NO
9      002000      MOV CHAN+SRCCHN,R0 ;GET CURRENT INPUT CHAN
10     002004      INC R0 ;MOVE TO NEXT CHAN
11     002006      CMP R0,#8. ;LAST CHAN?
12     002012      BHI 1$ ;YES, FLAG END OF INPUT
13     002014      CLR RECNUM+SRCCHN ;RESET RECORD (BLK) NUMBER
14     002020      MOV BLKTBL+<SRCCHN*4>,PTRTBL+<SRCCHN*4>
15     002026      MOV R0,CHAN+SRCCHN
16     002032      BIS #WINST,R0 ;CREATE A WAIT CALL FOR NEXT CHA
17     002036      MOV R0,0PC ;AND STORE IN NEXT LOCATION
18     002040      .WAIT 0
19     002042      BCS 1$ ;BRANCH IF NO MORE INPUT
20     002044      MOV #1,R0 ;FLAG END OF FILE
21     002050      RETURN 2$;
22     002052      MOV #1,R0 ;FLAG END OF INPUT
23     002056      RETURN

```

Figure 5-3  
Example of Page Heading from MACRO 80-Column Line Printer  
(same format as Terminal Listing)



# MACRO Assembler

RTEXEC RT-11 MACRO VM02-09 5-SEP-74 22:30:23 PAGE 29+  
SYMBOL TABLE

|                 |                     |                     |     |
|-----------------|---------------------|---------------------|-----|
| ABSEXP= ***** G | ARGCNT= ***** G     | ASSEM = ***** G     |     |
| BINCHN= 000004  | BINDAT 002322R      | 004 BLKTBL= 002310R | 004 |
| BPMB = 000020   | BUFTBL 000374RG     | 003 CHAN 002362R    | 004 |
| CHNSPC 000312R  | 003 CHRPN= ***** G  | CLK50 = 000040      |     |
| CMILEN= 000123  | CNTTBL 000360RG     | 003 CONFIG= 000300  |     |
| CONT 000040RG   | 010 CORERR 001726R  | 010 CPL = 000120    |     |
| CR = 000015     | CRFBUF 002076RG     | 004 CRFC = 000040   |     |
| CRFCHN= 000012  | CRFCNT 000004RG     | 007 CRFDAT 002352R  | 004 |
| CRFE = 000100   | CRFFLG 000000R      | 007 CRFLEN= 000204  |     |
| CRFM = 000010   | CRFP = 000020       | CRFPNT 000064R      | 003 |
| CRFR = 000004   | CRFS = 000002       | CRFSPC 000114R      | 003 |
| CRFTAB 000026R  | 003 CRFTST 000002RG | 007 CSIERR 000214R  | 003 |
| CTLTL 000000R   | 003 DATE 001000R    | 010 DATTIM 001004RG | 004 |
| DETEXT 000104R  | 003 DEVFUL 000252R  | 003 DIV60 001240R   | 010 |
| DNC = ***** G   | EDMASK= ***** G     | ED.ABS= ***** G     |     |
| EMTERR= 000052  | ENDP1 = ***** G     | ENDP2 = ***** G     |     |
| ENDSWT 000434R  | 010 ERR 001662R     | 010 ERRB 000102R    | 010 |
| ERRBTS= ***** G | ERRCNT= ***** G     | EXMFLG= ***** G     |     |
| FF = 000014     | FILNF 000264R       | 003 FIN 001434RG    | 010 |
| FINCL 001636R   | 010 FINMSG 001030R  | 004 FINMS1 001052R  | 004 |
| FINMS2 001070R  | 004 FINP1 000776R   | 010 FINP2 000776R   | 010 |
| FINSML 002124RG | 010 FRECOR 000006R  | 007 GETPLI 001756RG | 010 |
| GETR50= ***** G | GSARG = ***** G     | HDRTTL 001102RG     | 004 |
| HIGHAD= 000050  | ILLCMD 000226R      | 003 ILLDEV 000240R  | 003 |
| IMPUR 000042R   | 007 IMPURS 000000R  | 007 INIOF 000106R   | 010 |
| :               |                     |                     |     |
| :               |                     |                     |     |

RTEXEC RT-11 MACRO VM02-09 5-SEP-74 22:30:23 PAGE 29+  
SYMBOL TABLE

|                         |     |                  |     |                  |     |
|-------------------------|-----|------------------|-----|------------------|-----|
| TIME 000210R            | 003 | TIMTIM 001016R   | 004 | TIMWRD 000204R   | 003 |
| TPCNT= 000014           |     | TSTSTK 001704RG  | 010 | TTLBRK= ***** G  |     |
| TTLBUF= ***** G         |     | TTLLEN= 000040   |     | TTYBUF= 000616   |     |
| USRLOC= 000046          |     | VT = 000013      |     | WINST = 104240   |     |
| WRTERR 002306R          | 010 | XBAW = 000000    |     | XEDPIC= 000000   |     |
| XMIT0 = ***** G         |     | \$CLOUT 003006RG | 010 | \$EDABL= ***** G |     |
| \$FLUSH 002732RG        | 010 | \$NLIST= ***** G |     | \$READ 002422RG  | 010 |
| \$READW 002422RG        | 010 | \$WAIT 002730RG  | 010 | \$WRITE 002134RG | 010 |
| \$WRITW 002134RG        | 010 |                  |     |                  |     |
| . ABS. 000000           | 000 |                  |     |                  |     |
| 000000                  | 001 |                  |     |                  |     |
| DPURE 000000            | 002 |                  |     |                  |     |
| DPURES 000410           | 003 |                  |     |                  |     |
| MIXEDS 002376           | 004 |                  |     |                  |     |
| SWTSES 000000           | 005 |                  |     |                  |     |
| SWTSEC 000000           | 006 |                  |     |                  |     |
| IMPURS 000042           | 007 |                  |     |                  |     |
| MAINS 003024            | 010 |                  |     |                  |     |
| ERRORS DETECTED: 0      |     |                  |     |                  |     |
| FREE CORE: 13431. WORDS |     |                  |     |                  |     |

,LP:/C/L:BEX=RP4;RTPAR,RPARAM,RCIOCH,RTEXEC

Figure 5-4  
Symbol Table



## MACRO Assembler

5.5.1.2 Page Headings - The MACRO Assembler outputs each page in the format shown in Figure 5-3. On the first line of each listing page the assembler prints (from right to left):

1. title taken from .TITLE directive (most recent one encountered)
2. assembler version identification
3. the date and time of day if entered
4. page number

The second line of each listing page contains the subtitle text specified in the last encountered .SBTTL directive.

5.5.1.3 .TITLE - The .TITLE directive is used to print a heading in the output listing and to assign a name to the object module. The heading printed on the first line of each page of the listing is taken from the first 31 characters of the argument in the .TITLE directive. The first six characters (symbol name) of this same line are also used as the name of the object module. These six characters must be Radix-50 characters (any characters beyond the first six are ignored). Non Radix-50 characters are not acceptable.

For example:

```
.TITLE PROG TO PERFORM DAILY ACCOUNTING
```

causes PROG TO PERFORM DAILY ACCOUNTIN to be printed in the heading for each page and causes the object module of the assembled program to be PROG (this name is distinguished from the filename of the object module specified in the command string to the assembler).

If there is no TITLE statement, the default name assigned to the first object module is:

```
.MAIN.
```

The first tab or space following the .TITLE directive is not considered part of the object module name or header text, although subsequent tabs and spaces are significant.

If there is more than one .TITLE directive, the last .TITLE directive in the program conveys the name of the object module.

5.5.1.4 .SBTTL - The .SBTTL directive is used to provide the elements for a printed table of contents of the assembly listing. The text following the directive is printed as the second line of each of the following assembly listing pages until the next occurrence of a .SBTTL directive.

For example:

```
.SBTTL CONDITIONAL ASSEMBLIES
```



## MACRO Assembler

The text:

### CONDITIONAL ASSEMBLIES

is printed as the second line of each of the following assembly listing pages.

During pass 1 of the assembly process, MACRO automatically prints a table of contents for the listing containing the line sequence number and text of each .SBTTL directive in the program. Such a table of contents is inhibited by specifying the .NLIST TOC directive within the source.

An example of a table of contents is shown in Figure 5-5.

```
.MAIN, RT-11 MACRO VM02-09 5-SEP-74 22:30:23
TABLE OF CONTENTS

1- 29      RT-11 MACRO PARAMETER FILE
1- 37      COMMON PARAMETER FILE
2- 1       ASSEMBLY OPTIONS
3- 1       VARIABLE PARAMETERS
4- 1       GLOBALS
5- 1       SECTOR INITIALIZATION
7- 1       SUBROUTINE CALL DEFINITIONS
10- 1      MISCELLANEOUS MACRO DEFINITIONS
11- 2      MCIOCH - I/O CHANNEL ASSIGNMENTS
12- 2      ****EXEC****
13- 1      PROGRAM START
14- 1      INIT OUTPUT FILES
15- 1      SWITCH HANDLERS
16- 1      END-OF-PASS ROUTINES
17- 1      SWITCH AND DATE DATA AREAS
18- 1      INIT OUTPUT FILES (CONTINUED)
19- 1      FINISH ASSEMBLY AND RESTART
20- 1      MEMORY MANAGEMENT
21- 1      GET PHYSICAL SOURCE LINE
22- 1      SYSTEM MACRO HANDLERS
23- 1      WRITE ROUTINES
24- 1      READ ROUTINE
25- 1      COMMON I/O ROUTINES
26- 1      MESSAGES
27- 1      I/O TABLES
29- 1      FINIS
```

Figure 5-5  
Assembly Listing Table of Contents

Table of Contents text is taken from the text of each .SBTTL directive. The associated numbers are the page and line numbers of the .SBTTL directives.



## MACRO Assembler

5.5.1.5 .IDENT - The .IDENT directive is not used or supported by the RT-11 system, but is handled by MACRO for compatibility with other systems. .IDENT provides a means of labeling the object module produced as a result of a MACRO assembly. In addition to the name assigned to the object module with the .TITLE directive, a character string (up to six characters, treated like a .RAD50 string) can be specified between paired delimiters. For example:

```
.IDENT /V005A/
```

The character string:

```
V005A
```

is converted to Radix-50 notation and output to the global symbol directory of the object module.

When more than one .IDENT directive is found in a given program, the last .IDENT found determines the symbol which is passed as part of the object module identification.

5.5.1.6 Page Ejection (.PAGE Directive) - There are several means of obtaining a page eject in a MACRO assembly listing:

1. After a line count of 58 lines, MACRO automatically performs a page eject to skip over page perforations on line printer paper and to formulate terminal output into pages.
2. A form feed character used as a line terminator (or as the only character on a line) causes a page eject. Used within a macro definition a form feed character causes a page eject. A page eject is not performed when the macro is invoked.
3. More commonly, the .PAGE directive is used within the source code to perform a page eject at that point. The format of this directive is:

```
.PAGE
```

This directive takes no arguments and causes a skip to the top of the next page.

Used within a macro definition, the .PAGE is ignored, but the page eject is performed at each invocation of that macro.

## 5.5.2 Functions: .ENABL and .DSABL Directives

Several functions are provided by MACRO through the .ENABL and .DSABL directives. These directives use 3-character symbolic arguments to designate the desired function and are of the forms:

```
.ENABL arg  
.DSABL arg
```

where arg is one of the legal symbolic arguments defined below.



## MACRO Assembler

The following list describes the symbolic arguments and their associated functions in the MACRO language:

| <u>Symbolic<br/>Argument</u> | <u>Function</u>  |
|------------------------------|--|
| ABS                          | Enabling of this function (has no effect in ASEMBL) produces absolute binary output; (i.e., for input to the Paper Tape Software System absolute binary loader using a .BIN extension instead of .OBJ). The default case is .DSABL ABS.  |
| AMA                          | Enabling of this function directs the assembly of all relative addresses (address mode 67) as absolute addresses (address mode 37). This switch is useful during the debugging phase of program development.   |
| CDR                          | The statement .ENABL CDR (has no effect in ASEMBL) causes source columns 73 and greater to be treated as comments. This accommodates sequence numbers in card columns 72-80.   |
| FPT                          | Enabling of this function (has no effect in ASEMBL) causes floating point truncation, rather than rounding as is otherwise performed. .DSABL FPT returns to floating point rounding mode.  |
| LC                           | Enabling of this function causes the assembler to accept lower case ASCII input instead of converting it to upper case (has no effect in ASEMBL).  |
| LSB                          | Enable or disable a local symbol block (has no effect in ASEMBL). While a local symbol block is normally entered by encountering a new symbolic label or .CSECT directive, .ENABL LSB forces a local symbol block which is not terminated until a label of .CSECT directive following the .DSABL LSB statement is encountered. The default case is .DSABL LSB. |
| PNC                          | The statement .DSABL PNC (has no effect in ASEMBL) inhibits binary output until an .ENABL PNC is encountered. The default case is .ENABL PNC.  |

An incorrect argument causes the directive containing it to be flagged as an error.

### 5.5.3 Data Storage Directives

A wide range of data and data types can be generated with the following directives and assembly characters:



## MACRO Assembler

```
.BYTE  
.WORD  
"  
.ASCII  
.ASCIZ  
.RAD50  
↑B  
↑D  
↑O
```

These facilities are explained in the following sections.

5.5.3.1 .BYTE - The .BYTE directive is used to generate successive bytes of data. The directive is of the form:

```
.BYTE exp          ;WHICH STORES THE OCTAL  
                   ;EQUIVALENT OF THE EXPRESSION  
                   ;EXP IN THE NEXT BYTE  
  
.BYTE exp1,exp2,;WHICH STORES THE OCTAL  
                   ;EQUIVALENTS OF THE LIST OF  
                   ;EXPRESSIONS IN SUCCESSIVE BYTES.
```

A legal expression must have an absolute value (or contain a reference to an external symbol) and must result in eight bits or less of data. The 16-bit value of the expression must have a high-order byte (which is truncated) that is either all zeros or all ones. Each operand expression is stored in a byte of the object program. Multiple operands are separated by commas and stored in successive bytes. For example:

```
SAM=5  
;=.+410  
.BYTE "048,SAM      ;060 (OCTAL EQUIVALENT OF 48  
                   ;DECIMAL) IS STORED IN LOCATION  
                   ;411 + 005 IS STORED IN  
                   ;LOCATION 411
```

If the high-order byte of the expression equates to a value other than 0 or -1, it is truncated to the low-order eight bits and flagged with a T error code. If the expression is relocatable, an A-type warning flag is given.

At link time it is likely that relocation will result in an expression of more than eight bits, in which case, the Linker prints an error message. For example:



## MACRO Assembler

```
      .BYTE 23      ;STORES OCTAL 23 IN NEXT BYTE
B:    .BYTE B        ;RELOCATABLE VALUE CAUSES AN "A"
                        ;ERROR FLAG

      .GLOBL X
      X=3
      .BYTE X        ;STORES 3 IN NEXT BYTE
```

In the case where X is defined in another program:

```
      .GLOBL X
      .BYTE X
```

If an operand following the .BYTE directive is null, it is interpreted as a zero. For example:

```
      ., +420
      .BYTE ,,      ;ZEROS ARE STORED IN BYTES
                        ;420, 421, AND 422.
```

5.5.3.2 .WORD - The .WORD directive is used to generate successive words of data. The directive is of the form:

```
      .WORD exp      ;WHICH STORES THE OCTAL
                        ;EQUIVALENT OF THE EXPRESSION
                        ;EXP IN THE NEXT WORD

      .WORD exp1,exp2,... ;WHICH STORES THE OCTAL
                        ;EQUIVALENTS OF THE LIST OF
                        ;EXPRESSIONS IN SUCCESSIVE
                        ;WORDS
```

where a legal expression must result in 16 bits or less of data. Each operand expression is stored in a word of the object program.

Multiple operands are separated by commas and stored in successive words. For example:

```
      SAL=0
      ., +500
      .WORD 177535, +4, SAL ;STORES 177535, 506, AND 0
                        ;IN WORDS 500, 502, AND 504.
```

If an expression equates to a value of more than 16 bits, it is truncated and flagged with a T error code.

If an operand following the .WORD directive is null, it is interpreted as zero. For example:

```
      ., +500
      .WORD ,5,      ;STORES 0,5,0 IN LOCATIONS
                        ;500, 502, AND 504
```

A blank operator field (any operator not recognized as a macro call, op-code, directive or semicolon) is interpreted as an implicit .WORD directive. Use of this convention is discouraged. The first term of the first expression in the operand field must not be an instruction mnemonic or assembler directive unless preceded by a + or - operator.



## MACRO Assembler

For example:

```
      .R,440      ;THE OP-CODE FOR MOV, WHICH IS
LABEL: +MOV,LABEL ;1010000, IS STORED IN LOCATION
                  ;440, 440 IS STORED IN
                  ;LOCATION 442.
```

Note that the default .WORD directive occurs whenever there is a leading arithmetic or logical operator, or whenever a leading symbol is encountered which is not recognized as a macro call, an instruction mnemonic or assembler directive. Therefore, if an instruction mnemonic, macro call, or assembler directive is misspelled, the .WORD directive is assumed and errors will result. Assume that MOV is spelled incorrectly as MOR:

```
MOR    A,B
```

Two error codes result: A and U. Two words are then generated, one for MOR A and one for B.

5.5.3.3 ASCII Conversion of One or Two Characters - The ' and " characters are used to generate text characters within the source text. A single apostrophe followed by a character results in a term in which the 7-bit ASCII representation of the character is placed in the low-order byte and zero is placed in the high-order byte. For example:

```
MOV    #'A,R0
```

results in the following 16 bits being moved into R0:

```
0000000001000001
```

The ' character is never followed by a carriage return, null, RUBOUT, line feed, or form feed. (For another use of the ' character, see Section 5.6.3.6.)

STMNT:

```
GETSYM
BEQ    4S
CMPB   @CHRPNT, #' :    ;COLON DELIMITS LABEL FIELD
BEQ    LABEL
CMPB   @CHRPNT, #' =    ;EQUAL DELIMITS
BEQ    ASGMT            ;ASSIGNMENT PARAMETER
```

A double quote followed by two characters results in a term in which the 7-bit ASCII representations of the two characters are placed. For example:

```
MOV    #"AB,R0
```

results in the following binary word being moved into R0:

```
0100001001000001
```

Note that the first character is placed in the low-order byte and the second character in the high-order byte.



## MACRO Assembler

The " character is never followed by a carriage return, null, rubout, line feed, or form feed. For example:

```
DEVICE NAME TABLE
DEVNAM1 .WORD "RF      )RF DISK
          .WORD "RK      )RK DISK
DEVNKB1 .WORD "TT      )TERMINAL KEYBOARD
          .WORD "DT      )DECTAPE
          .WORD "LP      )LINE PRINTER
          .WORD "PR      )PAPER TAPE READER
          .WORD "PP      )PAPER TAPE PUNCH
          .WORD 0        )TABLE'S END
```

5.5.3.4 .ASCII - The .ASCII directive translates character strings into their 7-bit ASCII equivalents for use in the source program. The format of the .ASCII directive is:

.ASCII /character string/

where: character string is a string of any acceptable printing ASCII characters including spaces. The string may not include null characters, rubout, return, line feed, vertical tab, or form feed. Nonprinting characters can be expressed in digits of the current radix and delimited by angle brackets. (Any legal, defined expression is allowed between angle brackets.)

/ / are delimiting characters and may be any printing characters other than ; < and = characters and any character within the string.

As an example:

```
AS .ASCII /HELLO/ )STORES ASCII REPRESENTATION OF
                   )THE LETTERS H E L L O IN
                   )CONSECUTIVE BYTES
```

The order of the characters as they are stored in memory is illustrated below.



## MACRO Assembler

|      |   |   |      |
|------|---|---|------|
| 1001 |   |   | 1000 |
| 1003 | E | H | 1002 |
| 1005 | L | L | 1004 |
| 1007 |   | O | 1006 |
|      |   |   |      |

```
.ASCII /ABC/<15><12>/DEF/
      ;STORES
      /101,102,103,15,12,104,105,106
      ;IN CONSECUTIVE BYTES
```

```
.ASCII /<AB>/      ;STORES 74,101,102,76 IN
                  ;CONSECUTIVE BYTES
```

The ; and = characters are not illegal delimiting characters, but are preempted by their significance as a comment indicator and assignment operator, respectively. For other than the first group, semicolons are treated as beginning a comment field. For example:

|        | <u>Directive</u> | <u>Result</u> | <u>Explanation</u>  |
|--------|------------------|---------------|---|
| .ASCII | ;ABC;/DEF/       | A B C D E F   | Acceptable, but not recommended procedure.  |
| .ASCII | /ABC/;DEF;       | A B C         | ;DEF; is treated as a comment and ignored.  |
| .ASCII | /ABC/=DEF=       | A B C D E F   | Acceptable, but not recommended procedure.  |
| .ASCII | =DEF=            |               | The assignment .ASCII=DEF is performed and an error generated upon encountering the second =. |

5.5.3.5 .ASCIZ - The .ASCIZ directive is equivalent to the .ASCII directive with a zero byte automatically inserted as the final character of the string. For example:

When a list or text string has been created with a .ASCIZ directive, a search for the null character can determine the end of the list as follows:

```
CR=15
LF=12
```

```
.
```

```
MOV     #HELLO,R1
```



# MACRO Assembler

```

      MOV      #LINBUF,R2
X:    MOV      (R1)+,(R2)+      ;MOVE A CHARACTER OF THE
                                ;MESSAGE STRING INTO THE
                                ;OUTPUT BUFFER
      BNE      X               ;BRANCH BACK IF BYTE
                                ;NOT EQUAL TO 0
      .
      .
      .

HELLO: .ASCIZ  <CR><LF>/MACRO-11 V001A/<CR><LF>
                                ;INTRO MESSAGE

```

5.5.3.6 .RAD50 - The .RAD50 directive allows the user the capability to handle symbols in Radix-50 coded form (this form is sometimes referred to as MOD40 and is used in PDP-11 system programs). Radix-50 form allows three characters to be packed into sixteen bits; therefore, any 6-character symbol can be held in two words. The form of the directive is:

.RAD50 /string/

where: / / delimiters can be any printing characters other than the =, <, and ; characters.

string is a list of the characters to be converted (three characters per word) and may consist of the characters A through Z, 0 through 9, dollar (\$), dot (.) and space ( ). If there are fewer than three characters (or if the last set is fewer than three characters) they are considered to be left justified and trailing spaces are assumed. Illegal nonprinting characters are replaced with a ? character and cause an I error flag to be set. Illegal printing characters set the Q error flag.

The trailing delimiter may be a carriage return, semicolon, or matching delimiter. (A warning code is printed if it is not a matching delimiter, however.) For example:

```

***** A
20 00040 003223      .RAD50  /ABC      ;PACK ABC INTO ONE WORD
21                                     ;PACK AB (SPACE) INTO ONE WORD.
22 00042 003220      .RAD50  /AB/      ;PACK THREE SPACES INTO ONE WORD
23 00044 000000      .RAD50  //
24 00046 003223      .RAD50  /ABCD/    ;PACK ABC INTO FIRST WORD AND
    00050 014400                                     ;D (SPACE)(SPACE) INTO SECOND WORD

```

Each character is translated into its Radix-50 equivalent as indicated:



## MACRO Assembler

| <u>Character</u> | <u>Radix-50<br/>Equivalent (octal)</u> | <u>ASCII (octal)</u> |
|------------------|--|----------------------|
| (space)          | 0                                      | 40                   |
| A-Z              | 1-32                                   | 101-132              |
| \$               | 33                                     | 44                   |
| .                | 34                                     | 56                   |
| undefined        | 35                                     | undefined            |
| 0-9              | 36-47                                  | 60-71                |

Note that another character could be defined for code 35, which is currently unused.

The Radix-50 equivalents for three characters (C1,C2,C3) are combined in one 16-bit word as follows:

$$\text{Radix-50 value} = ((C1*50)+C2)*50+C3$$

For example:

Radix-50 value of ABC is  $((1*50)+2)*50+3$  or 3223

See Appendix E for a table to quickly determine Radix-50 equivalents.

Use of angle brackets is encouraged in the .ASCII, .ASCIIZ, and .RAD50 statements whenever leaving the text string to insert special codes. For example:

```
.ASCII <101> ;EQUIVALENT TO .ASCII/A/
.RAD50 /AB/<35>;STORES 3255 IN NEXT WORD.
CHR1=1
CHR2=2
CHR3=3
.
.
.
.RAD50 <CHR1><CHR2><CHR3> ;EQUIVALENT TO RAD50/ABC/
```

### 5.5.4 Radix Control

#### 5.5.4.1 .RADIX

Numbers used in a MACRO source program are initially considered to be octal numbers. However, the programmer has the option of declaring the following radices:

2, 4, 8, 10

This is done via the .RADIX directive of the form:

```
.RADIX n
```

where n is one of the acceptable radices.



The default radix at the start of each program, and the argument assumed if none is specified, is 8 (octal). For example:

In general it is recommended that macro definitions not contain or rely on radix settings from the .RADIX directive. The temporary radix control characters should be used within a macro definition. (↑D, ↑O, and ↑B are described in the following section.) A given radix is valid throughout a program until changed. Where a possible conflict exists within a macro definition or in possible future uses of that code module, it is suggested that the user specify values using the temporary radix controls.

MACRO has three unary operators to provide a single interpretation in a given radix within another radix as follows:

**For example:**

Notice that while the uparrow and radix specification characters may not be separated, the radix operator can be physically separated from the number by spaces or tabs for formatting purposes. Where a term or expression is to be interpreted in another radix, it should be enclosed in angle brackets.

A temporary radix change from octal to decimal may be made by specifying a decimal radix number with a "decimal point". For example:



## MACRO Assembler

```
100.      (144(8))
1376.     (2540(8))
128.      (200(8))
```

### 5.5.5 Location Counter Control

The four directives that control movement of the location counter are .EVEN and .ODD which move the counter a maximum of one byte, and .BLKB and .BLKW which allow the user to specify blocks of a given number of bytes or words to be skipped in the assembly.

5.5.5.1 .EVEN - The .EVEN directive ensures that the assembly location counter contains an even memory address by adding one if the current address is odd. If the assembly location counter is even, no action is taken. Any operands following a .EVEN directive are ignored.

The .EVEN directive is used as follows:

```
.ASCIZ /THIS IS A TEST/
.EVEN                                ;ASSURES NEXT STATEMENT
                                      ;BEGINS ON A WORD BOUNDARY
.WORD XYZ
```

5.5.5.2 .ODD - The .ODD directive ensures that the assembly location counter is odd by adding one if it is even. For example:

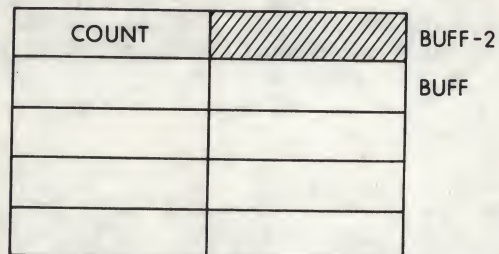
```
;CODE TO MOVE DATA FROM AN INPUT LINE
;TO A BUFFER

      N=5                          ;BUFFER HAS 5 WORDS
      .
      .
      .
      .ODD
      .BYTE N*2                    ;COUNT=2N BYTES
BUFF: .BLKW N                      ;RESERVE BUFFER OF N WORDS
      .
      .
      MOV #BUFF,R2                 ;ADDRESS OF EMPTY BUFFER IN R2
      MOV #LINE,R1                 ;ADDRESS OF INPUT LINE IS IN R1
      MOVB -1(R2),R0               ;GET COUNT STORED IN BUFF-1 IN R0
AGAIN: MOVB (R1)+,(R2)+             ;MOVE BYTE FROM LINE INTO BUFFER
      BEQ DONE                     ;WAS NULL CHARACTER SEEN?
      DEC R0                       ;DECREMENT COUNT
      BNE AGAIN                   ;NOT=0, GET NEXT CHARACTER
      .
      .
      CLRB -(R2)                   ;OUT OF ROOM IN BUFFER, CLEAR LAST
DONE:                                ;WORD
      .
      .
LINE: .ASCIZ /TEXT/
```



## MACRO Assembler

In this case, .ODD is used to place the buffer byte count in the byte preceding the buffer, as follows:



5.5.5.3 .BLKB and .BLKW - Blocks of storage can be reserved using the .BLKB and .BLKW directives. .BLKB is used to reserve byte blocks and .BLKW reserves word blocks. The two directives are of the form:

```
.BLKB    exp
.BLKW    exp
```

where exp is the number of bytes or words to reserve. If no argument is present, 1 is the assumed default value. Any legal expression which is completely defined at assembly time and produces an absolute number is legal. For example:

```
1
2
3
4      000000'      ,CSECT IMPURE
5 000000      PASS: .BLKW
6
7 000002      SYMBOL: .BLKW 2      ;NEXT GROUP MUST STAY TOGETHER
8 000006      MODE:      ;SYMBOL ACCUMULATOR
9 000006      FLAGS: .BLKB 1      ;FLAG BITS
10 00007      SECTOR: .BLKB 1      ;SYMBOL/EXPRESSION TYPE
11 00010      VALUE: .BLKW 1      ;EXPRESSION VALUE
12 00012      RELLVL: .BLKW 1
13           .BLKW 2      ;END OF GROUPED DATA
14 00020      CLCNAM: .BLKW 2      ;CURRENT LOCATION COUNTER
15 00024      CLCFGs: .BLKB 1
16 00025      CLCSEC: .BLKB 1
17 00026      CLCLOC: .BLKW 1
18 00030      CLCMAX: .BLKW 1
19      000001'      .END
```

The .BLKB directive has the same effect as:

```
.=.+exp
```

but is easier to interpret in the context of source code.

## 5.5.6 Numeric Control

Several directives are available to provide software complements to the floating-point hardware on the PDP-11.



## MACRO Assembler

A floating-point number is represented by a string of decimal digits. The string (which can be a single digit in length) may optionally contain a decimal point, and may be followed by an optional exponent indicator; in the form of the letter E and a signed decimal exponent. The list of number representations below contains seven distinct, valid representations of the same floating-point number:

```
3
3.
3.0
3.0E0
3E0
.3E1
300E-2
```

As can be quickly inferred, the list could be extended indefinitely (e.g., 3000E-3, .03E2, etc.). A leading plus sign is ignored (e.g., +3.0 is considered to be 3.0). Leading minus signs complement the sign bit. No other operators are allowed (e.g., 3.0+N is illegal).

Floating-point number representations are valid only in the contexts described in the remainder of this section.

Floating-point numbers are normally rounded. That is, when a floating-point number exceeds the limits of the field in which it is to be stored, the high-order excess bit is added to the low-order retained bit. For example, if the number were to be stored in a 2-word field, but more than 32 bits were needed for its value, the highest bit carried out of the field would be added to the least significant position. In order to enable floating-point truncation, the .ENABL FPT directive is used and .DSABL FPT is used to return to floating-point rounding.

5.5.6.1 .FLT2 and .FLT4 - Like the .WORD directive, the two floating-point storage directives cause their arguments to be stored in-line with the source program (have no effect in ASEMBL). These two directives are of the form:

```
.FLT2    arg1,arg2,...
.FLT4    arg1,arg2,...
```

where arg1,arg2, etc. represent one or more floating point numbers separated by commas.

.FLT2 causes two words of storage to be generated for each argument while .FLT4 generates four words of storage.



## MACRO Assembler

The following code was assembled with the 4-word floating-point math package:

```

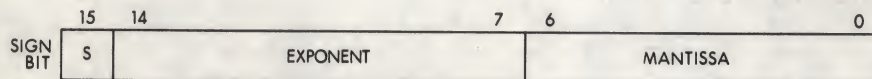
1
2
3
4 000000 037314 ATOFTB: .FLT4 1.E-1      110*-1
   000002 146314
   000004 146314
   000006 146315
5 000010 036443      .FLT4 1.E-2      110*-2
   000012 153412
   000014 036560
   000016 121727
6 000020 034721      .FLT4 1.E-4      110*-4
   000022 133427
   000024 054342
   000026 014545
7 000030 031453      .FLT4 1.E-8      110*-8
   000032 146167
   000034 010604
   000036 060717
8 000040 022746      .FLT4 1.E-16     110*-16
   000042 112624
   000044 137304
   000046 046741
9 000050 005517      .FLT4 1.E-32     110*-32
   000052 130436
   000054 126505
   000056 034625

```

5.5.6.2 Temporary Numeric Control:  $\uparrow F$  and  $\uparrow C$  - Like the temporary radix control operators, operators are available to specify either a 1-word floating-point number ( $\uparrow F$ --not available in ASEMBL) or the one's complement of a 1-word number ( $\uparrow C$ ). For example:

FL3.7:  $\uparrow F3.7$

creates a 1-word floating-point number at location FL3.7 containing the value 3.7 as follows:



This 1-word floating-point number is the first word of the 2- or 4-word floating-point number format shown in the PDP-11 PROCESSOR HANDBOOK, and the statement:

CMP151:  $\uparrow C151$

stores the one's complement of 151 in the current radix (assume current radix is octal) as follows:



## MACRO Assembler



Since these control operators are unary operators, their arguments may be integer constants or symbols, and the operators may be expressed successively. For example:

$\uparrow C \uparrow D25$  or  $\uparrow C31$  or 177746

The term created by the unary operator and its argument is then a term which can be used by itself or in an expression. For example:

$\uparrow C2+6$

is equivalent to:

$\langle \uparrow C2 \rangle +6$  or 177775+6 or 000003

For this reason, the use of angle brackets is advised. Expressions used as terms or arguments of a unary operator must be explicitly grouped.

An example of the importance of ordering with respect to unary operators is shown below:

|                   |          |
|-------------------|----------|
| $\uparrow F1.0$   | = 040200 |
| $\uparrow F-1.0$  | = 140200 |
| $-\uparrow F1.0$  | = 137600 |
| $-\uparrow F-1.0$ | = 037600 |

The argument to the  $\uparrow F$  operator must not be an expression and should be of the same format as arguments to the .FLT2 and .FLT4 directives.

### 5.5.7 Terminating Directives

5.5.7.1 .END - The .END directive indicates the physical end of the source program. The .END directive is of the form:

.END exp

where exp is an optional argument which, if present, indicates the program entry point, i.e., the transfer address.



## MACRO Assembler

When the load module is loaded, program execution begins at the transfer address indicated by the .END directive. In a runtime system (the load module output of the Linker) a .END exp statement should terminate the first object module and .END statements should terminate any other object modules.

5.5.7.2 .EOT - Under the RT-11 System, the .EOT directive is ignored. The physical end file allows several physically separate tapes to be assembled as one program.

### 5.5.8 Program Boundaries Directive: .LIMIT

The .LIMIT directive reserves two words into which the Linker puts the low and high addresses of the load module's relocatable code (the load module is the result of the link). The low address (inserted into the first word) is the address of the first byte of code. The high address is the address of the first free byte following the relocated code. These addresses are always even since all relocatable sections are loaded at even addresses. (If a relocatable section consists of an odd number of bytes, the Linker adds one to the size to make it even.)

### 5.5.9 Program Section Directives

The assembler provides for 255(10) program sections: an absolute section declared by .ASECT, an unnamed relocatable program section declared by .CSECT, and 253(10) named relocatable program sections declared by .CSECT symbol, where symbol is any legal symbolic name. These directives allow the user to:

1. Create his program (object module) in sections:

The assembler maintains separate location counters for each section. This allows the user to write statements which are not physically contiguous but will be loaded contiguously. The following examples will clarify this:



# MACRO Assembler

```

      .CSECT          ISTART THE UNNAMED RELOCATABLE SECTION
A:    0              IASSEMBLED AT RELOCATABLE 0,
B:    0              IRELOCATABLE 2 AND
C:    0              IRELOCATABLE 4
ST:   CLR A          IASSEMBLE CODE AT
      CLR B          IRELOCATABLE ADDRESS
      CLR C          I6 THROUGH 21
      .ASECT         ISTART THE ABSOLUTE SECTION
      .B4            IASSEMBLE CODE AT
      .WORD ,+2,HALT IABSOLUTE 4 THROUGH 7
      .CSECT         IRESUME THE UNNAMED RELOCATABLE
                      ISECTION
      INC A          IASSEMBLE CODE AT
      BR ST          IRELOCATABLE 22 THROUGH 27
      .END

```

The first appearance of .CSECT or .ASECT assumes the location counter is at relocatable or absolute zero, respectively. The scope of each directive extends until a directive to the contrary is given. Further occurrences of the same .CSECT or .ASECT resume assembling where the section was left off.

```

      .CSECT COM1      IDECLARE SECTION COM1
A:    0              IASSEMBLED AT RELOCATABLE 0
B:    0              IASSEMBLED AT RELOCATABLE 2
C:    0              IASSEMBLED AT RELOCATABLE 4
      .CSECT COM2      IDECLARE SECTION COM2
X:    0              IASSEMBLED AT RELOCATABLE 0
Y:    0              IASSEMBLED AT RELOCATABLE 2
      .CSECT COM1      IRETURN TO COM1
D:    0              IASSEMBLED AT RELOCATABLE 6
      .END

```

The assembler automatically begins assembling at relocatable zero of the unnamed .CSECT if not instructed otherwise; that is, the first statement of an assembly is an implied .CSECT.

All labels in an absolute section are absolute; all labels in a relocatable section are relocatable. The location counter symbol, ".", is relocatable or absolute when referenced in a relocatable or absolute section, respectively. Undefined internal symbols are assigned the value of relocatable or absolute zero in a relocatable or absolute section, respectively. Any labels appearing on a .ASECT or .CSECT statement are assigned the value of the location counter before the .ASECT or .CSECT takes effect. Thus, if the first statement of a program is:

```
A:    .ASECT
```

then A is assigned to relocatable zero and is associated with the unnamed relocatable section (because the assembler implicitly begins assembly in the unnamed relocatable section).

Since it is not known at assembly time where the program sections are to be loaded, all references between sections in a single assembly are translated by the assembler to references relative to the base of that section. The assembler provides the Linker with the necessary information to resolve the linkage. Note that this is not necessary when



## MACRO Assembler

making a reference to an absolute section (the assembler knows all load addresses of an absolute section).

Examples:

```
      .ASECT
      .#1000
AI    CLR X          ;ASSEMBLED AS CLR BASE OF UNNAMED
      ;RELOCATABLE SECTION +12
      JMP Y          ;ASSEMBLED AS JMP BASE OF UNNAMED
      ;RELOCATABLE SECTION + 10

      .CSECT
      MOV R0,R1
      JMP A          ;ASSEMBLED AS JMP 1000
Y:    HALT
X:    0
      .END
```

In the above example the references to X and Y were translated into references relative to the base of the unnamed relocatable section.

2. Share code and/or data between object modules (separate assemblies):

Named relocatable program sections operate as FORTRAN labeled COMMON; that is, sections of the same name from different assemblies are all loaded at the same location by LINK. The unnamed relocatable section is the exception to this as all unnamed relocatable sections are loaded in unique areas by LINK.

Note that there is no conflict between internal symbolic names and program section names; that is, it is legal to use the same symbolic name for both purposes. In fact, considering FORTRAN again, this is necessary to accommodate the FORTRAN statement:

```
COMMON  /X/A,B,C,X
```

where the symbol X represents the base of this program section and also the fourth element of this program section.

Program section names should not duplicate .GLOBL names. In FORTRAN language, COMMON block names and SUBROUTINE names should not be the same.

The .ASECT and .CSECT program section directives are provided in MACRO to allow the user to specify an unnamed absolute or relocatable section. These directives are formatted as follows:

```
.ASECT
.CSECT
.CSECT symbol
```



## MACRO Assembler

The single absolute section can be declared with an:

`.ASECT`

directive. No name can be associated with the absolute section specified by means of the `.ASECT` directive. The single unnamed relocatable program section can be declared with a:

`.CSECT`

directive.

All named relocatable sections are loaded in unique areas by LINK. Up to 253(10) named relocatable program sections can be declared with:

`.CSECT symbol`

directives, where symbol is any legal symbolic name.

The assembler automatically begins assembling at relocatable zero of the unnamed `.CSECT` if not instructed otherwise; that is, the first statement of an assembly is an implied `.CSECT`.

### 5.5.10 Symbol Control: `.GLOBL`

If a program is created in segments which are assembled separately, global symbols are used to allow reference to one symbol by the different segments.

A global symbol must be declared in a `.GLOBL` directive. The form of the `.GLOBL` directive is:

`.GLOBL sym1,sym2,...`

where:

`sym1,sym2, etc.` are legal symbolic names, separated by commas, tabs, or spaces where more than one symbol is specified.

Symbols appearing in a `.GLOBL` directive are either defined within the current program or are external symbols, in which case they are defined in another program which is to be linked with the current program, by LINK, prior to execution.

A `.GLOBL` directive line may contain a label in the label field and comments in the comment field.

At the end of assembly pass 1, MACRO has determined whether a given global symbol is defined within the program or is expected to be an external symbol.



## MACRO Assembler

DEFINE A SUBROUTINE WITH 2 ENTRY POINTS WHICH  
CALLS AN EXTERNAL SUBROUTINE

```
      .CSECT          ;DECLARE THE CONTROL SECTION
      .GLOBL  A,B,C  ;DECLARE A,B,C AS GLOBALS
A:    MOV    0(R5)+,R0 ;ENTRY A IS DEFINED
      MOV    #X,R1
X:    JSR    PC,C      ;CALL EXTERNAL SUBROUTINE C
      RTS    R5        ;EXIT
B:    MOV    0(R5)+,R1 ;DEFINE ENTRY B
      CLR    R1
      BR     X
```

In the previous example, A and B are entry symbols (entry points), C is an external symbol and X is an internal symbol.

A global symbol is defined only when it appears in a .GLOBL directive. A symbol is not considered a global symbol if it is assigned the value of a global expression in a direct assignment statement.

References to external symbols can appear in the operand field of an instruction or assembler directive in the form of a direct reference, i.e.:

```
CLR    EXT
.WORD  EXT
CLR    0EXT
```

or a direct reference plus or minus a constant, i.e.:

```
D=6
CLR    EXT+0
.WORD  EXT-2
CLR    0EXT+0
```

An external symbol cannot be used in the evaluation of a direct assignment expression. A global symbol defined within the program can be used in the evaluation of a direct assignment statement.

### 5.5.11 Conditional Assembly Directives

Conditional assembly directives provide the programmer with the capability to conditionally include or ignore blocks of source code in the assembly process. This technique is used extensively to allow several variations of a program to be generated from the source program.

The general form of a conditional block is as follows:

```
.IF  cond,argument(s) ;START CONDITIONAL BLOCK
      .                ;STATEMENTS IN RANGE OF
      .                ;CONDITIONAL
      .                ;BLOCK
.ENDC                    ;END CONDITIONAL BLOCK
```

where: cond is a condition which must be met if the block is to be included in the assembly. These conditions are defined below.



## MACRO Assembler

argument(s) are a function of the condition to be tested. If more than one argument is specified, they must be separated by commas.

range is the body of code which is included in the assembly or ignored depending upon whether the condition is met.

The following are the allowable conditions:

| Conditions |            | Arguments                                     | Assemble Block If                  |
|------------|------------|---|------------------------------------|
| Positive   | Complement |   |                                    |
| EQ         | NE         | expression                                    | expression=0 (or $\neq$ 0)         |
| GT         | LE         | expression                                    | expression>0 (or $\leq$ 0)         |
| LT         | GE         | expression                                    | expression<0 (or $\geq$ 0)         |
| DF         | NDF        | symbolic argument                             | symbol is defined (or undefined)   |
| B          | NB         | macro-type argument                           | argument is blank (or nonblank)    |
| IDN        | DIF        | two macro-type arguments separated by a comma | arguments identical (or different) |
| Z          | NZ         | expression                                    | same as EQ/NE                      |
| G          |            | expression                                    | same as GT/LE                      |
| L          |            | expression                                    | same as LT/GE                      |

IF DIF and IF IDN are not available in ASEMBL.

### NOTE

A macro-type argument is enclosed in angle brackets or within an up-arrow construction (as described in Section 5.2.1.1). For example:

<A,B,C>  
↑/124/

For example:

```

ALPHA=1
      .IF      EQ,ALPHA+1      /ASSEMBLE IF ALPHA+1=0
      .
      .
      .ENDC

```

Within the conditions DF and NDF the following two operators are allowed to group symbolic arguments:

&            logical AND operator  
!            logical inclusive OR operator



## MACRO Assembler

For example:

```
.IF      DF,SYM1 & SYM2  ;ASSEMBLE IF BOTH SYM1
:                                     ;AND SYM2 ARE DEFINED
:
.ENDC
```

5.5.11.1 Subconditionals - Subconditionals may be placed within conditional blocks to indicate:

1. assembly of an alternate body of code when the condition of the block indicates that the code within the block is not to be assembled,
2. assembly of a non-contiguous body of code within the conditional block depending upon the result of the conditional test to enter the block,
3. unconditional assembly of a body of code within a conditional block.

There are three subconditional directives, as follows:

| <u>Subconditional</u> | <u>Function</u>   |
|-----------------------|---|
| .IFF                  | The code following this statement up to the next subconditional or end of the conditional block is included in the program if the value of the condition tested upon entering the conditional block is false.       |
| .IFT                  | The code following this statement up to the next subconditional or end of the conditional block is included in the program if the value of the condition tested upon entering the conditional block is true.        |
| .IFTF                 | The code following this statement up to the next subconditional or the end of the conditional block is included in the program regardless of the value of the condition tested upon entering the conditional block. |

The implied argument of the subconditionals is the value of the condition upon entering the conditional block. Subconditionals are used within outer level conditional blocks. Subconditionals are ignored within nested, unsatisfied conditional blocks.

For example:



## MACRO Assembler

```
.IF      DF,SYM  /ASSEMBLE BLOCK IF SYM IS DEFINED
.IFF                                /ASSEMBLE THE FOLLOWING CODE ONLY IF
                                   /SYM IS UNDEFINED
.
.IFT                                /ASSEMBLE THE FOLLOWING CODE ONLY IF
                                   /SYM IS DEFINED
.
.IFTF                               /ASSEMBLE THE FOLLOWING CODE
                                   /UNCONDITIONALLY
.
.
.ENDC

.IF      DF,X    /ASSEMBLY TESTS FALSE
.IF      DF,Y    /TESTS FALSE
.IFF                                /NESTED CONDITIONAL
                                   /IGNORED WITHIN NESTED, UNSATISFIED
                                   /CONDITIONAL BLOCK
.
.
.IFT                                /NOT SEEN
.
.ENDC
.ENDC
```

However,

```
.IF      DF,X    /TESTS TRUE
.IF      DF,Y    /TESTS FALSE
.IFF                                /IS ASSEMBLED
                                   /OUTER CONDITIONAL SATISFIED.
.
.IFT                                /NOT ASSEMBLED
.
.ENDC
.ENDC
```

5.5.11.2 Immediate Conditionals - An immediate conditional directive is a means of writing a 1-line conditional block. In this form, no .ENDC statement is required and the condition is completely expressed on the line containing the conditional directive. Immediate conditions are of the form:

.IIF cond, arg, statement

where: cond is one of the legal conditions defined for conditional blocks in Section 5.5.11.



## MACRO Assembler

arg is the argument associated with the conditional specified, that is, either an expression, symbol, or macro-type argument, as described in Section 5.5.11.

statement is the statement to be executed if the condition is met.

For example:

```
.IIF DF,FOO,BEQ ALPHA
```

This statement generates the code:

```
BEQ ALPHA
```

if the symbol FOO is defined.

A label must not be placed in the label field of the .IIF statement. Any necessary labels may be placed on the previous line, as in the following example:

```
LABEL: .IIF DF,FPP BEQ ALPHA
```

or included as part of the conditional statement:

```
.IIF DF,FOO LABEL: BEQ ALPHA
```

5.5.11.3 PAL-11R and PAL-11S Conditional Assembly Directives - In order to maintain compatibility with programs developed under PAL-11R and PAL-11S, the following conditionals remain permissible under MACRO. It is advisable that future programs be developed using the format for MACRO conditional assembly directives.

| <u>Directive</u> | <u>Arguments</u>   | <u>Assemble Block if</u>       |
|------------------|--------------------|--------------------------------|
| .IFZ or .IFEQ    | expression         | expression=0                   |
| .IFNZ or .IFNE   | expression         | expression#0                   |
| .IFL or .IFLT    | expression         | expression<0                   |
| .IFG or .IFGT    | expression         | expression>0                   |
| .IFGE            | expression         | expression=>0                  |
| .IFLE            | expression         | expression<=0                  |
| .IFDF            | logical expression | expression is true(defined)    |
| .IFNDF           | logical expression | expression is false(undefined) |

The rules governing the usage of these directives are now the same as for the MACRO conditional assembly directives previously described. Conditional assembly blocks must end with the .ENDC directive and are limited to a nesting depth of 16(10) levels (instead of the 127(10) levels allowed under PAL-11R).



## MACRO Assembler

### 5.6 MACRO DIRECTIVES

#### 5.6.1 Macro Definition

It is often convenient in assembly language programming to generate a recurring coding sequence with a single statement. In order to do this, the desired coding sequence is first defined with dummy arguments as a macro. Once a macro has been defined, a single statement calling the macro by name with a list of real arguments (replacing the corresponding dummy arguments in the definition) generates the correct sequence or expansion.

5.6.1.1 **.MACRO** - The first statement of a macro definition must be a **.MACRO** directive (not available in **ASEMBL**). The **.MACRO** directive is of the form:

**.MACRO** name, dummy argument list

where:

|                     |  |
|---------------------|--|
| name                | is the name of the macro. This name is any legal symbol. The name chosen may be used as a label elsewhere in the program.  |
| ,                   | represents any legal separator (generally a comma or space).   |
| dummy argument list | zero, one, or more legal symbols which may appear anywhere in the body of the macro definition, even as a label. These symbols can be used elsewhere in the user program with no conflicts of definition. Where more than one dummy argument is used, they are separated by any legal separator (generally a comma). |

A comment may follow the dummy argument list in a statement containing a **.MACRO** directive. For example:

```
.MACRO  ABS      A,B      ;DEFINE MACRO ABS WITH TWO ARGUMENTS
```

A label must not appear on a **.MACRO** statement. Labels are sometimes used on macro calls, but serve no function when attached to **.MACRO** statements.

5.6.1.2 **.ENDM** - The final statement of every macro definition must be an **.ENDM** directive (not available in **ASEMBL**) of the form:

```
.ENDM name
```

where name is an optional argument and is the name of the macro being terminated by the statement.

For example:



## MACRO Assembler

`.ENDM` (terminates the current macro definition)

`.ENDM ABS` (terminates the definition of the macro ABS)

If specified, the symbolic name in the `.ENDM` statement must correspond to that in the matching `.MACRO` statement. Otherwise the statement is flagged and processing continues. Specification of the macro name in the `.ENDM` statement permits the assembler to detect missing `.ENDM` statements or improperly nested macro definitions.

The `.ENDM` statement may contain a comment field, but must not contain a label.

An example of a macro definition is shown below:

```
.MACRO  TYPMSG  MESSAGE  ;TYPE A MESSAGE
JSR     R5,TYPMSG
.WORD   MESSAGE
.ENDM
```

5.6.1.3 `.MEXIT` - In order to implement alternate exit points from a macro (particularly nested macros), the `.MEXIT` directive is provided. `.MEXIT` (not available in `ASEMBL`) terminates the current macro as though an `.ENDM` directive were encountered. Use of `.MEXIT` bypasses the complications of conditional nesting and alternate paths. For example:

```
.MACRO  ALTR      N,A,B
.
.
. IF      EQ,N      ;START CONDITIONAL BLOCK
.
.
. MEXIT          ;EXIT FROM MACRO DURING CONDITIONAL
                  ;BLOCK
. ENDC          ;END CONDITIONAL BLOCK
.
.
. ENDM          ;NORMAL END OF MACRO
```

In an assembly where `N=0`, the `.MEXIT` directive terminates the macro expansion.

Where macros are nested, a `.MEXIT` causes an exit to the next higher level. A `.MEXIT` encountered outside a macro definition is flagged as an error.

5.6.1.4 Macro Definition Formatting - A form feed character used as a line terminator in a macro source statement (or as the only character on a line), causes a page eject when the source program is listed. Used within a macro definition, a form feed character also causes a page eject. A page eject is not performed, however, when the macro is invoked.



## MACRO Assembler

Used within a macro definition, the .PAGE directive is ignored, but a page eject is performed at invocation of that macro.

### 5.6.2 Macro Calls

A macro must be defined prior to its first reference. Macro calls are of the general form:

|        |   |
|--------|---|
| label: | name, real arguments  |
| where: | label represents an optional statement label.   |
|        | name represents the name of the macro specified in the .MACRO directive preceding the macro definition.   |
|        | represents any legal separator (comma, space, or tab). No separator is necessary where there are no real arguments. (Refer to Section 5.2.1.1.)   |
|        | real arguments are those symbols, expressions, and values which replace the dummy arguments in the .MACRO statement. Where more than one argument is used, they are separated by any legal separator. |

Where a macro name is the same as a user label, the appearance of the symbol in the operation field designates a macro call, and the occurrence of the symbol in the operand field designates a label reference. For example:

```
ABS:  MOV    R0,R1      ;ABS IS USED AS A LABEL
      .
      .
      BR     ABS        ;ABS IS CONSIDERED A LABEL
      .
      ABS    #4,ENT,LAR  ;CALL MACRO WITH 3 ARGUMENTS
```

Arguments to the macro call are treated as character strings whose usage is determined by the macro definition.

### 5.6.3 Arguments to Macro Calls and Definitions

Arguments within a macro definition or macro call are separated from other arguments by any of the separating characters described in Section 5.2.1.1. For example:

```
.MACRO REN    A,B,C    ;MACRO DEFINITION
.
.
.
REN    ALPHA,BETA,<C1,C2>    ;MACRO CALL
```



## MACRO Assembler

Arguments which contain separating characters are enclosed in paired angle brackets. An up-arrow construction is provided to allow angle brackets to be passed as arguments.

For example:

```
REN      <MOV      X,Y>,#44,WEV
```

This call would cause the entire statement:

```
MOV      X,Y
```

to replace all occurrences of the symbol A in the macro definition. Real arguments within a macro call are considered to be character strings and are treated as a single entity until their use in the macro expansion.

The up-arrow construction could have been used in the above macro call as follows:

```
REN      ^/MOV      X,Y/,#44,WEV
```

which is equivalent to:

```
REN      <MOV      X,Y>,#44,WEV
```

Since spaces are ignored preceding an argument, they can be used to increase legibility of bracketed constructions.

5.6.3.1 Macro Nesting - Macro nesting (nested macro calls), where the expansion of one macro includes a call to another macro, causes one set of angle brackets to be removed from an argument with each nesting level. The depth of nesting allowed is dependent upon the amount of memory space used by the program. To pass an argument containing legal argument delimiters to nested macros, the argument should be enclosed in one set of angle brackets for each level of nesting, as shown below:

```
R0=X0
R1=X1
X=10

.MACRO LEVEL1 DUM1,DUM2
LEVEL2 DUM1
LEVEL2 DUM2
.ENDM
.MACRO LEVEL2 DUM3
DUM3
ADD      #10,R0
MOV      R0,(R1)+
.ENDM
```

A call to the LEVEL1 macro:

```
LEVEL1 <<MOV      X,R0>>,<<CLR      R0>>
```

causes the following expansion:



## MACRO Assembler

```
MOV    X,R0
ADD    #10,R0
MOV    R0,(R1)+
CLR    R0
ADD    #10,R0
MOV    R0,(R1)+
```

Where macro definitions are nested (that is, a macro definition is entirely contained within the definition of another macro) the inner definition is not defined as a callable macro until the outer macro has been called and expanded. For example:

```
.MACRO LV1    A,B
:
:
.MACRO LV2    A
:
:
.ENDM
.ENDM
```

The LV2 macro cannot be called by name until after the first call to the LV1 macro. Likewise, any macro defined within the LV2 macro definition cannot be referenced directly until LV2 has been called.

5.6.3.2 Special Characters - Arguments may include special characters without enclosing the argument in a bracket construction if that argument does not contain spaces, tabs, semicolons, or commas. For example:

```
.MACRO PUSH    ARG
MOV    ARG,=(SP)
.ENDM
:
:
:
PUSH    X+3(X2)
```

generates the following code:

```
MOV    X+3(X2),=(SP)
```

5.6.3.3 Numeric Arguments Passed as Symbols - When passing macro arguments, a useful capability is to pass a symbol which can be treated by the macro as a numeric string. An argument preceded by the unary operator backslash (\) is treated as a number in the current radix. (\ is not available in ASEMBL.) The ASCII characters representing the number are inserted in the macro expansion; their function is defined in context (see Section 5.6.3.6 for an explanation of single-quote usage). For example:



## MACRO Assembler

```

A'B: .MACRO CNT A,B
      .WORD
      .ENDM
      C=0
      .MACRO INC A,B
      CNT A,\B
      B=B+1
      .ENDM
      .
      .
      INC X,C

```

The macro call would expand to:

```
X0: .WORD
```

A subsequent identical call to the same macro would generate:

```
X1: .WORD
```

and so on for later calls. The two macros are necessary because the dummy value of B cannot be updated in the CNT macro. In the CNT macro, the number passed is treated as a string argument. (Where the value of the real argument is 0, a single 0 character is passed to the macro expansion.)

The number being passed can also be used to make source listings somewhat clearer. For example, versions of programs created through conditional assembly of a single source can identify themselves as follows:

```

.MACRO IOT SYM ;ASSUME THAT THE SYMBOL ID TAKES
.IDENT /SYM/ ;ON A UNIQUE 2 DIGIT VALUE FOR
.ENDM IOT ;EACH POSSIBLE CONDITIONAL
;ASSEMBLY OF THE PROGRAM

.MACRO OUT ARG
IOT 005A'ARG
.ENDM
.
.
.
OUT \ID ;WHERE 005A IS THE UPDATE VERSION
;OF THE PROGRAM AND ARG INDICATES
;THE CONDITIONAL ASSEMBLY VERSION

```

The above macro call expands to:

```
.IDENT /005AXX/
```

where XX is the conditional value of ID.

Two macros are necessary since the text delimiting characters in the .IDENT statement would inhibit the concatenation of a dummy argument.



## MACRO Assembler

5.6.3.4 Number of Arguments - If more arguments appear in the macro call than in the macro definition, the excess arguments are ignored. If fewer arguments appear in the macro call than in the definition, missing arguments are assumed to be null (consist of no characters). The conditional directives .IF B and .IF NB can be used within the macro to detect null arguments.

A macro can be defined with no arguments.

5.6.3.5 Automatically Created Symbols Within User-Defined Macros - MACRO can be made to create symbols of the form n\$ where n is a decimal integer number such that  $64 < n < 127$ . Created symbols are always local symbols between 64\$ and 127\$. Such local symbols are created by the assembler in numerical order, i.e.:

```
64$
65$
.
.
.
126$
127$
```

Created symbols are particularly useful where a label is required in the expanded macro. Such a label must otherwise be explicitly stated as an argument with each macro call or the same label is generated with each expansion (resulting in a multiply-defined label). Unless a label is referenced from outside the macro, there is no reason for the programmer to be concerned with that label.

The range of these local symbols extends between two explicit labels. Each new explicit label causes a new local symbol block to be initialized.

The macro processor creates a local symbol on each call of a macro whose definition contains a dummy argument preceded by the ? character. For example:

```
      .MACRO  ALPHA  A,?B
      TST     A
      BEQ     B
      ADD     #5,A
B:
      .ENDM
```

Local symbols are generated only where the real argument of the macro call is either null or missing. If a real argument is specified in the macro call, the generation of a local symbol is inhibited and normal replacement is performed. Consider the following expansions of the macro ALPHA above.



## MACRO Assembler

Generate a local symbol for missing argument:

```
ALPHA    X1
TST      X1
BEQ      648
ADD      #5,X1
```

648:

Do not generate a local symbol:

```
ALPHA    X2,XYZ
TST      X2
BEQ      XYZ
ADD      #5,X2
```

XYZ:

These assembler-generated symbols are restricted to the first sixteen (decimal) arguments of a macro definition.

5.6.3.6 Concatenation - The apostrophe or single quote character (') operates as a legal separating character in macro definitions. An ' character which precedes and/or follows a dummy argument in a macro definition is removed, and the substitution of the real argument occurs at that point. For example:

```
      .MACRO DEF      A,B,C
A'B:  .ASCIZ  /C/
      .WORD   'A''B
      .ENDM
```

When this macro is called:

```
DEF      X,Y,<MACRO-11>
```

it expands as follows:

```
XY:      .ASCIZ  /MACRO-11/
          .WORD   'X'Y
```

In the macro definition, the scan terminates upon finding the first ' character. Since A is a dummy argument, the ' is removed. The scan resumes with B, notes B as another dummy argument and concatenates the two dummy arguments. The third dummy argument is noted as going into the operand of the .ASCIZ directive. On the next line (this example is for purely illustrative purposes) the argument to .WORD is seen as follows: The scan begins with a ' character. Since it is neither preceded nor followed by a dummy argument, the ' character remains in the macro definition. The scan then encounters the second ' character which is followed by a dummy argument and is discarded. The scan of the argument A terminates upon encountering the second ' which is also discarded since it follows a dummy argument. The next ' character is neither preceded nor followed by a dummy argument and remains in the macro expansion. The last ' character is followed by another dummy argument and is discarded. (Note that the five ' characters were necessary to generate two ' characters in the macro expansion.)

Within nested macro definitions, multiple single quotes can be used, with one quote removed at each level of macro nesting.



## MACRO Assembler

### 5.6.4 .NARG, .NCHR, and .NTYPE

These three directives allow the user to obtain the number of arguments in a macro call (.NARG), the number of characters in an argument (.NCHR), or the addressing mode of an argument (.NTYPE). (They are not available in ASEMBL.) Use of these directives permits selective modifications of a macro depending upon the nature of the arguments passed.

The .NARG directive enables the macro being expanded to determine the number of arguments supplied in the macro call and is of the form:

label: .NARG symbol

where: label is an optional statement label

symbol is any legal symbol which is equated to the number of arguments in the macro call currently being expanded. The symbol can be used by itself or in expressions.

This directive can occur only within a macro definition. An example of the use of .NARG follows.

```
.MACRO  ARGS      A1,A2,A3,A4
.NARG   NUM
.IF EQ  NUM=1     ;IF A2, A3, A4 WERE
                ;NOT SPECIFIED
.WORD   A1
.IFF                      ;IF ALL ARGS WERE GIVEN
.WORD   A1
.ASCII  /A2/
.WORD   A3
.ASCII  /A4/
.ENDC
.ENDM

ARGS     ALPHA
.WORD    ALPHA                      generated

ARGS     ALPHA,BETA,GAMMA,DELTA
.WORD    ALPHA
.ASCII   /BETA/                      generated
.WORD    GAMMA
.ASCII   /DELTA/
```

The .NCHR directive enables a program to determine the number of characters in a character string, and is of the form:

label: .NCHR symbol, <character string>

where: label is an optional statement label

symbol is any legal symbol which is equated to the number of characters in the specified character string. The symbol is separated from the character string argument by any legal separator.



## MACRO Assembler

<character string>

is a string of printing characters which should only be enclosed in angle brackets if it contains a legal separator. A semicolon also terminates the character string.

This directive can occur anywhere in a MACRO program. For example:

```
.MACRO CHARS A
.NCHR NUM,A
.ASCII /A/
.IF EQ NUM&1          ;IF THE STRING CONTAINS
                        ;AN EVEN NUMBER OF
                        ;CHARACTERS

.WORD -1
.IFF                  ;IF STRING LENGTH IS ODD
.BYTE -2
.ENDC
.ENDM
```

For example, using the above definition, the code:

CHARS ALPHA

expands to:

```
.ASCII /ALPHA/
.BYTE -2
```

and

CHARS BETA

expands to:

```
.ASCII /BETA/
.WORD -1
```

The .NTYPE directive enables the macro being expanded to determine the addressing mode and register of any argument, and is of the form:

label: .NTYPE symbol, arg

where: label is an optional statement label

symbol is any legal symbol, the low order 6-bits of which is equated to the 6-bit addressing mode of the argument. The symbol is separated from the argument by a legal separator. This symbol can be used by itself or in expressions.

arg is any legal macro argument (dummy argument) as defined in Section 5.6.3.

This directive can occur only within a macro definition. An example of .NTYPE usage in a macro definition is shown below:



## MACRO Assembler

```
.MACRO SAVE ARG
.NTYPE SYM,ARG
.IF EQ,SYM&70
MOV ARG,TEMP ;REGISTER MODE
.IFF
MOV #ARG,TEMP ;NON-REGISTER MODE
.ENDC
.ENDM
```

Using this definition, the code:

```
SAVE R2
```

expands to:

```
MOV R2,TEMP
```

and

```
SAVE ALPHA
```

expands to:

```
MOV ALPHA,TEMP
```

### 5.6.5 .ERROR and .PRINT

The .ERROR directive (not available in ASEMBL) is used to output messages to the listing file during assembly pass 2. A common use is to provide diagnostic announcements of a rejected or erroneous macro call. The form of the .ERROR directive is as follows:

```
label: .ERROR expr;text
```

where: label is an optional statement label

expr is an optional legal expression whose value is output to the listing file when the .ERROR directive is encountered. Where expr is not specified, the text only is output to the listing file.

; denotes the beginning of the text string to be output.

text is the string to be output to the listing file. The text string is terminated by a line terminator.

Upon encountering a .ERROR directive anywhere in a MACRO program, the assembler outputs a single line containing:

1. the sequence number of the .ERROR directive line,
2. the current value of the location counter,
3. the value of the expression if one is specified, and,



## MACRO Assembler

4. the text string specified.

For example, assume the following error macro occurs:

```
.MACRO STORE SRC,DEST
.NTYPE A,DEST
.IF EQ,<A&7>=6 ;IF STACK POINTER USED
.ERROR          A;UNACCEPTABLE MACRO ARGUMENT
.IFF
MOV             SRC,DEST
.ENDC
.ENDM
```

```
STORE R3,04(SP)
```

and the following line is output:

```
***** P
00000 000076          .ERROR          A;UNACCEPTABLE MACRO ARGUMENT
```

This message is used to indicate an inability of the subject macro to cope with the argument DEST which is detected as being indexed deferred addressing mode (mode 70) with the stack pointer (%6) used as the index register. The line is flagged on the assembly listing with a P error code.

The .PRINT directive is identical to .ERROR except that it is not flagged with a P error code. (.PRINT is not available in ASEMBL.)

### 5.6.6 Indefinite Repeat Block: .IRP and .IRPC

An indefinite repeat block (not available in ASEMBL) is a structure very similar to a macro definition. An indefinite repeat is essentially a macro definition which has only one dummy argument and is expanded once for every real argument supplied. An indefinite repeat block is coded in-line with its expansion rather than being referenced by name as a macro is referenced. An indefinite repeat block is of the form:

```
label:      .IRP arg,<real arguments>
            .
            .
            (range of the indefinite repeat)
            .
            .
            .ENDM
```

where: label is an optional statement label. A label may not appear on any .IRP statement within another macro definition, repeat range or indefinite repeat range, or on any .ENDM statement.

arg is a dummy argument which is successively replaced with the real arguments in the .IRP statement.



## MACRO Assembler

### <real arguments>

is a list of arguments to be used in the expansion of the indefinite repeat range and enclosed in angle brackets. Each real argument is a string of zero or more characters or a list of real arguments (enclosed in angle brackets). The real arguments are separated by commas.

### range

is the code to be repeated once for each real argument in the list. The range may contain macro definitions, repeat blocks, or other indefinite repeat blocks. Note that only created symbols should be used as labels within an indefinite repeat range.

An indefinite repeat block can occur either within or outside macro definitions, repeat ranges, or indefinite repeat ranges. The rules for creating an indefinite repeat block are the same as for the creation of a macro definition (for example, the .MEXIT statement is allowed in an indefinite repeat block). Indefinite repeat arguments follow the same rules as macro arguments. A second type of indefinite repeat block is available which handles character substitution rather than argument substitution. The .IRPC directive is used as follows:

```
label:  .IRPC arg,string
        .
        .
        (range of indefinite repeat)
        .
        .
        .ENDM
```

On each iteration of the indefinite repeat range, the dummy argument (arg) assumes the value of each successive character in the string. Terminators for the string are: space, comma, tab, carriage return, line feed, and semicolon.

Figure 5-6 is an example of .IRP and .IRPC usage.

IRPTST RT-11 MACRO VM02-09 11:04:49 PAGE 1

```
1
2
3
4
5
6      .TITLE  IRPTST
7      .LIST   MD,MC,ME
8      R0=X0
9 000000 012700  MOV    #TABLE,R0
      000056'
```



## MACRO Assembler

```

10      ,IRP      X,<A,B,C,D,E,F>
11
12      MOV      X,(R0)+
13      ,ENDM

      00004 016720      MOV      A,(R0)+
              000032

      00010 016720      MOV      B,(R0)+
              000030

      00014 016720      MOV      C,(R0)+
              000026

      00020 016720      MOV      D,(R0)+
              000024

      00024 016720      MOV      E,(R0)+
              000022

      00030 016720      MOV      F,(R0)+
              000020

14      ,IRPC     X,ABCDEF
15      ,ASCII    /X/
16      ,ENDM

      00034      101      ,ASCII    /A/
      00035      102      ,ASCII    /B/
      00036      103      ,ASCII    /C/
      00037      104      ,ASCII    /D/
      00040      105      ,ASCII    /E/
      00041      106      ,ASCII    /F/

17
18 00042 041101 A:      ,WORD      "AB
19 00044 041502 B:      ,WORD      "BC
20 00046 042103 C:      ,WORD      "CD
21 00050 042504 D:      ,WORD      "DE
22 00052 043105 E:      ,WORD      "EF
23 00054 043506 F:      ,WORD      "FG
24 00056      TABLE1 ,BLKW      6
25      000001'      ,END

```

Figure 5-6  
.IRP and .IRPC Example

### 5.6.7 Repeat Block: .REPT

Occasionally it is useful to duplicate a block of code a number of times in line with other source code. (.REPT is not available in ASEMBL.) This is performed by creating a repeat block of the form:

```

label:      .REPT expr
            .
            .
            .
            (range of repeat block)
            .

```



## MACRO Assembler

```
.  
.  
.ENDM      ;OR .ENDR
```

where: label is an optional statement label. The .ENDR or .ENDM directive may not have a label. A .REPT statement occurring within another repeat block, indefinite repeat block, or macro definition may not have a label associated with it.

expr is any legal expression controlling the number of times the block of code is assembled. Where  $\text{expr} < 0$ , the range of the repeat block is not assembled.

range is the code to be repeated expr number of times. The range may contain macro definitions, indefinite repeat blocks, or other repeat blocks. Note that no statements within a repeat range can have a label.

The last statement in a repeat block can be an .ENDM or .ENDR statement. The .ENDR statement is provided for compatibility with previous assemblers.

The .MEXIT statement is also legal within the range of a repeat block.

### 5.6.8 Macro Libraries: .MCALL

All macro definitions must occur prior to their referencing within the user program. MACRO provides a selection mechanism for the programmer to indicate in advance those system macro definitions required by his program.

The .MCALL directive is used to specify the names of all system macro definitions not defined in the current program but required by the program (not available in ASEMBL). The .MCALL directive must appear before the first occurrence of a macro call for an externally defined macro. The .MCALL directive is of the form:

```
.MCALL arg1,arg2,...
```

where arg1,arg2, etc. are the names of the macro definitions required in the current program.

When this directive is encountered, MACRO searches the system library file SYSMAC.SML, to find the requested definition(s). MACRO searches for SYSMAC.SML on the system device.

See Appendix D for a listing of the system macro file (SYSMAC.SML) stored on the system device.

## 5.7 CALLING AND USING MACRO

The MACRO Assembler assembles one or more ASCII source files containing MACRO statements into a single relocatable binary object file. Assembler output consists of this binary object file and an



## MACRO Assembler

optional assembly listing followed by the symbol table listing. CREF (Cross Reference) listings may also be specified as part of the assembly output by means of switch options.

MACRO is executed using the RT-11 Monitor R command as follows:

.R MACRO

The assembler responds by typing an asterisk (\*) to indicate readiness to accept command string input. In response to the \* printed by the assembler, the user types the output file specification(s), followed by an equal sign or left angle bracket, followed by the input file specification(s) in a command line as follows:

\*dev:obj,dev:list/s:arg=dev:sourcel,...,dev:sourcen/s:arg

|        |                         |   |
|--------|-------------------------|---|
| where: | dev:                    | is any legal RT-11 device   |
|        | obj                     | is the binary object file   |
|        | list                    | is the assembly listing file containing the assembly listing and symbol table                               |
|        | sourcel,<br>...,sourcen | are the ASCII source files containing the macro source program(s); a maximum of six source files is allowed |
|        | /s:arg                  | represents a switch and argument as explained in Section 5.7.1  |

A null specification in either of the output file fields signifies that the associated output file is not desired.

One or more switches can be indicated with the appropriate file specification to provide MACRO with information about that file.

The default case for each file specification is noted below:

| <u>file</u>             | <u>device</u>                       | <u>filename</u> | <u>extension</u> |
|-------------------------|-------------------------------------|-----------------|------------------|
| object                  | DK:                                 | -               | .OBJ             |
| listing                 | device used<br>for object<br>output | -               | .LST             |
| sourcel                 | DK:                                 | -               | .MAC             |
| source2                 | device used                         | -               | .MAC             |
| .                       | for last source                     |                 |                  |
| .                       | file specified                      |                 |                  |
| .                       |                                     |                 |                  |
| sourcen                 |                                     |                 |                  |
| system<br>macro<br>file | system device<br>SY:                | SYSMAC          | .SML             |

Type CTRL C to halt MACRO at any time and return control to the monitor. To restart the assembler type R MACRO or the REENTER command in response to the monitor's dot.



## MACRO Assembler

### NOTE

If ↑C was typed while a CREF listing was being produced, the REENTER command may not be accepted. In this case, type R MACRO to restart the assembler.

#### 5.7.1 Switches

There are three types of switch options: listing control switches, function switches, and CREF specification switches. The listing control switches (/L,/N) provide capabilities similar to those described in detail in section 5.5.1.1. The function control switches (/D,/E) provide function control as described in Section 5.5.2; arguments for these switches are summarized in Section 5.7.1.2. CREF control switches allow the user to obtain a detailed cross-referenced listing of his assembled file, and are described in detail in Section 5.7.1.3. Multiple arguments may be specified for a particular switch, if desired, by separating each switch value from the next by a colon. For example:

/N:TTM:CND

These switches turn off teleprinter mode and suppress printing of unsatisfied conditionals (as described in the next section). Also, the switches are not restricted to appearing near a particular file in the command string; /N:TTM, for example, is legal in all of the following places:

\* ,LP:/N:TTM=source  
\* ,LP:=source/N:TTM  
\* /N:TTM,LP:=source

and they are all equivalent in function.

**5.7.1.1 Listing Control Switches** - A listing control switch (/L for list or /N for nolist) is indicated in a command line as follows:

\*dev:obj.ext,dev:list.ext/s:arg=dev:source.ext

where s:arg represents /L or /N; the remainder of the command line abbreviations are as described in Section 5.7.

The /N switch with no argument causes only the symbol table and error listings to be produced. The /L switch with no arguments causes .LIST and .NLIST directives that appear in the source program but have no arguments to be ignored. A summary of the arguments which are valid for the listing control switches follows (refer to Section 5.5.1.1 for details):

| <u>Argument</u> | <u>Default</u> | <u>Controls listing of</u>   |
|-----------------|----------------|------------------------------|
| SEQ             | list           | Source line sequence numbers |
| LOC             | list           | Location counter             |
| BIN             | list           | Generated binary code        |
| BEX             | list           | Binary extensions            |



## MACRO Assembler

|     |               |  |
|-----|---------------|--|
| SRC | list          | Source code  |
| COM | list          | Comments   |
| MD  | list          | Macro definitions, repeat range expansions         |
| MC  | list          | Macro calls, repeat range expansions               |
| ME  | nolist        | Macro expansions                                   |
| MEB | nolist        | Macro expansion binary code                        |
| CND | list          | Unsatisfied conditionals, .IF and .ENDC statements |
| LD  | nolist        | Listing directives with no arguments               |
| TOC | list          | Table of Contents                                  |
| TTM | terminal mode | Listing output format                              |
| SYM | list          | Symbol table                                       |

For example, a command line in the following format allows binary code to be listed throughout the assembly using the 132-column line printer format:

```
* ,LP:/L:MEB/N:TTM=FILE
```

5.7.1.2 Function Switches - The function control switches (/D for disable and /E for enable) are used to enable or disable certain functions in source input files and are indicated in the command line as follows:

```
*dev:obj.ext,dev:list.ext=dev:source/s:arg
```

/s:arg here represents either /D:arg or /E:arg. A summary of the arguments which are valid for use with the function control switches follows (refer to Section 5.5.2 for details):

| <u>Argument</u> | <u>Default</u> | <u>Enables or disables</u>                               |
|-----------------|----------------|--|
| ABS             | disable        | Absolute binary output                                   |
| AMA             | disable        | Assembly of all absolute addresses as relative addresses |
| CDR             | disable        | Source columns 73 and greater to be treated as comments  |
| FPT             | disable        | Floating point truncation                                |
| LC              | disable        | Accepts lower case ASCII input                           |
| LSB             | disable        | Local symbol lock  |
| PNC             | enable         | Binary output  |

The following command line allows all 80 columns of a card to be used as input:

```
* ,LP:=CR:/E:CDR
```

Use of either the function control or listing control switches and arguments at assembly-time will override any corresponding listing or function control directives and arguments in the source program. For example, assume the following appears in the source program:

```
.NLIST MEB
:
:      MACRO References
:
.LIST MEB
```



## MACRO Assembler

The "MEB" printing will be disabled for the block indicated; however, if /L:MEB is indicated in the assembly command line, both the .NLIST MEB and the .LIST MEB will be ignored and the "MEB" printing will be enabled everywhere in the program.

5.7.1.3 Cross Reference Table Generation (CREF) - A cross reference table of all or a subset of all symbols used in the source program and the statements where they were defined or used can be obtained automatically following an assembly by specifying /C:arg with the assembly listing file specification (and any listing or function control specifications) as follows:

```
*dev:obj.ext,dev:list.ext/s:arg/C:arg=dev:source.ext
```

/s:arg represents /L:arg, /N:arg, /E:arg, or /D:arg.

There are six sections to a complete cross reference listing:

1. Cross reference of program symbols (i.e., labels used in the program and symbols used on the left of the "=" operator).
2. Cross reference of register-equate symbols (those symbols which are defined in the program by a "SYMBOL=%N",  $0 \leq N \leq 7$ , construct. (Normally this consists of the symbols R0, R1, R2, R3, R4, R5, SP, and PC.)
3. Cross reference of MACRO symbols (names of macros as defined by a .MACRO directive, or as specified in a .MCALL directive).
4. Cross reference of permanent symbols (all operation mnemonics and assembler directives).
5. Cross reference of control sections (those names specified as the operand of a .CSECT directive, plus the blank .CSECT and the absolute section ". ABS." which are always defined by MACRO).
6. Cross reference of errors (all errors flagged on the listing are grouped by error type).

Any or all of the above sections may be included in the cross reference listing as desired. The associated switch options and their arguments are listed below:

| <u>Switch<br/>Argument</u> | <u>Section Type</u>                             |
|----------------------------|---|
| /C:S                       | User-defined symbols                            |
| /C:R                       | Register symbols                                |
| /C:M                       | Macro symbolic names                            |
| /C:P                       | Permanent symbols<br>(instructions, directives) |
| /C:C                       | Control sections (.CSECT<br>symbolic names)     |
| /C:E                       | Error codes                                     |
| /C<no arg>                 | Equivalent to /C:S:M:E                          |



## MACRO Assembler

The specification of a /C switch in a command string causes a temporary file, "DK:CREF.TMP", to be generated. If device DK: is write-locked or contains insufficient free space for the temporary file, the user may allocate the temporary file on another device. To do so, a third output file specification is given in the MACRO command string; this file is then used instead of DK:CREF.TMP, and is purged after use. For example, a command string of this type:

```
* ,LP:,RK2:TEMP.TMP=SOURCE/C
```

causes "RK2:TEMP.TMP" to be used as the temporary file.

Figure 5-7 illustrates assembled source code and Figure 5-8 contains the CREF output. The command line used to produce these listings was:

```
* ,LP:/C:S:H:R:P:C:E/N:BEX=EXAMPL
```

An explanation of the CREF output follows the figures.



EXAMPLE OF CROSS-REFERENCE LIST RT-11 MACRO VM02-09 5-SEP-74 22:11:59 PAGE 1

```

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39
.TITLE EXAMPLE OF CROSS-REFERENCE LISTING
;DEFINE THE REGISTER SYMBOLS
X0
X1
X2
X3
X4
X5
X6
X7
012
000000 R0=
000001 R1=
000002 R2=
000003 R3=
000004 R4=
000005 R5=
000006 SP=
000007 PC=
000012 LF=

.MCALL .TTYIN, .EXIT
.MACRO CALL NAME
JSR PC,NAME
.ENDM

.GLOBAL SUBR1, SUBR2

.CSECT
MOV #BUFFER,R2
.TTYIN
MOVB R0,(R2)+
CMPB R0,#LF
BNE 1$
CLRB (R2)+
MOV #BUFFER,R3
CALL SUBR1
RCS START
CALL SUBR2
MOV R0,ANSWER
.EXIT
ANSWER: .BLKW
BUFFER: .BLKB
72.

000000' .END
000000' START

```

Figure 5-7  
MACRO Source Code



EXAMPLE OF CROSS-REFERENCE LIST RT-11 MACRO VM02-09 5-SEP-74 22:11:59 PAGE 1+

SYMBOL TABLE

```

ANSWER 000046R      002  BUFFER 000050R      002  LF      = 000012
PC      =X0000007      R0      =X0000000      R1      =X0000001
R2      =X0000002      R3      =X0000003      R4      =X0000004
R5      =X0000005      SP      =X0000006      START  000000R      002
SUBR1 = ***** G
. ABS. 000000      000
      000000      001
PROG  000160      002
ERRORS DETECTED: 0
FREE CORE: 16248. WORDS

,LP:/CIS:MIR:PIC:E/NIBEX=EXAMPL

```

Figure 5-7 (Cont.)  
MACRO Source Code

EXAMPLE OF CROSS-REFERENCE LIST RT-11 MACRO VM02-09 5-SEP-74 22:11:59 PAGE 5-1  
CROSS REFERENCE TABLE (CREF VM1-02 )

```

ANSWER 1-24      1-36#
BUFFER 1-33*      1-29
LF      1-12#      1-26
START  1-23#      1-31
SUBR1  1-20#      1-30
SUBR2  1-20#      1-32

```

Figure 5-8  
CREF Listing Output



EXAMPLE OF CROSS-REFERENCE LIST RT-11 MACRO VM02-09 5-SEP-74 22:11:59 PAGE R-1  
CROSS REFERENCE TABLE (CREF V01-02 )

|    |       |       |       |       |
|----|-------|-------|-------|-------|
| PC | 1-10# | 1-30* | 1-32* |       |
| R0 | 1-3#  | 1-25  | 1-26  | 1-33  |
| R1 | 1-4#  |       |       |       |
| R2 | 1-5#  | 1-23* | 1-25* | 1-28* |
| R3 | 1-6#  | 1-29* |       |       |
| R4 | 1-7#  |       |       |       |
| R5 | 1-8#  |       |       |       |
| SP | 1-9#  |       |       |       |

EXAMPLE OF CROSS-REFERENCE LIST RT-11 MACRO VM02-09 5-SEP-74 22:11:59 PAGE M-1  
CROSS REFERENCE TABLE (CREF V01-02 )

|        |       |      |      |
|--------|-------|------|------|
| .EXIT  | 1-14# | 1-34 |      |
| .TTYIN | 1-14# | 1-24 |      |
| CALL   | 1-16# | 1-30 | 1-32 |

Figure 5-8 (Cont.)  
CREF Listing Output



EXAMPLE OF CROSS-REFERENCE LIST RT-11 MACRO VM02-09 5-SEP-74 22:11:59 PAGE P-1  
CROSS REFERENCE TABLE (CREF V01-02 )

|        |      |      |
|--------|------|------|
| .BLKR  | 1-37 |      |
| .BLKW  | 1-36 |      |
| .CSECT | 1-22 |      |
| .END   | 1-39 |      |
| .GLOBL | 1-20 |      |
| .IF    | 1-24 |      |
| .MACRO | 1-16 |      |
| .MCALL | 1-14 |      |
| .TITLE | 1-1  |      |
| .RCS   | 1-24 | 1-31 |
| .BNE   | 1-27 |      |
| .CLRB  | 1-28 |      |
| .CMPB  | 1-26 |      |
| .EMT   | 1-24 | 1-34 |
| .JSR   | 1-30 | 1-32 |
| .MOV   | 1-23 | 1-29 |
| .MOVB  | 1-25 |      |
|        |      | 1-33 |

EXAMPLE OF CROSS-REFERENCE LIST RT-11 MACRO VM02-09 5-SEP-74 22:11:59 PAGE C-1  
CROSS REFERENCE TABLE (CREF V01-02 )

|       |      |
|-------|------|
| .ABS. | 0-0  |
| .PRNG | 0-0  |
|       | 1-22 |

Figure 5-8 (Cont.)  
CREF Listing Output



## MACRO Assembler

Cross reference tables, if requested, are generated at the end of a MACRO assembly listing. Each table begins on a new page (the tables in Figure 5-8 have been consolidated due to space considerations). Symbols, control sections, and error codes are listed at the left margin of the page; corresponding references are indicated next to them across the page from left to right. A reference is of the form p-1, where p is the page on which the symbol, control section, or error code appears, and 1 is the line number within the page. A number sign (#) appears next to a reference wherever a symbol has been defined. An asterisk appears next to a reference wherever a destructive reference has been made to the symbol (i.e., the contents of the location defined by that symbol has been altered at that point).

The CREF output requested in the preceding figures included user defined symbols, macro symbolic names, control sections, error codes, register symbols, and permanent symbols. Since no errors were generated in this assembly, no CREF output for error codes was produced.

### 5.8 ERROR MESSAGES

MACRO error messages enclosed in question marks are output on the terminal. The single-letter error codes are printed in the assembly listing.

In terminal mode these error codes are printed following a field of six asterisk characters and on the line preceding the source line containing the error. For example:

```
***** A
26 00236 000002' .WORD REL1+REL2
```

| <u>Error Code</u> | <u>Meaning</u>   |
|-------------------|--|
| A                 | Addressing error. An address within the instruction is incorrect. Also may indicate a relocation error. The addition of two relocatable symbols is flagged as an A error. May also indicate that a local symbol is being defined more than 128 words from the beginning of a local symbol block. |
| B                 | Bounding error. Instructions or word data would be assembled at an odd address in memory. The location counter is updated by +1.   |
| D                 | Doubly-defined symbol referenced. Reference was made to a symbol which is defined more than once.  |
| E                 | End directive not found. (A .END is generated.)  |
| I                 | Illegal character detected. Illegal characters which are also non-printing are replaced by a ? on the listing. The character is then ignored.  |
| L                 | Line buffer overflow, i.e., input line greater than 132 characters. Extra characters on a line, (more than 72 (10)) are ignored in terminal mode.  |



## MACRO Assembler

|   |   |
|---|---|
| M | Multiple definition of a label. A label was encountered which was equivalent (in the first six characters) to a previously encountered label.                               |
| N | Number containing 8 or 9 has decimal point missing.   |
| O | Opcode error. Directive out of context.   |
| P | Phase error. A label's definition of value varies from one pass to another.   |
| Q | Questionable syntax. There are missing arguments or the instruction scan was not completed or a carriage return was not immediately followed by a line feed or form feed.   |
| R | Register-type error. An invalid use of or reference to a register has been made.  |
| T | Truncation error. A number generated more than 16 bits of significance or an expression generated more than 8 bits of significance during the use of the .BYTE directive.   |
| U | Undefined symbol. An undefined symbol was encountered during the evaluation of an expression. Relative to the expression, the undefined symbol is assigned a value of zero. |
| Z | Instruction which is not compatible among all members of the PDP-11 family (11/05, 11/20, 11/40, 11/45).  |

| <u>Error Message</u>     | <u>Explanation</u>  |
|--------------------------|---|
| ?BAD SWITCH?             | The switch specified was not recognized by the program.   |
| ?INSUFFICIENT CORE?      | There are too many symbols in the program being assembled. Try dividing program into separately-assembled subprograms.  |
| ?I/O ERROR ON CHANNEL n? | <p>A hardware error occurred while attempting to read from or write to the device on the channel specified in the message. (Channel numbers <math>0 \leq n \leq 10</math> octal) are assigned to files in the manner described in Section 9.4.7, Chapter 9.)</p> <p>Note that the CREF temporary file is on channel 2 even if it was not specified in the command string (i.e., if the default file DK:CREF.TMP is used).</p> |
| ?NO INPUT FILE?          | No input file was specified and there must be at least one input file.  |



## MACRO Assembler

?OUTPUT DEVICE FULL?

No room to continue writing output.  
Try to compress device with PIP.

All CREF error messages begin with C- to distinguish them from MACRO error messages. When a CREF error occurs, the error message is printed on the console terminal and CREF chains back to MACRO; MACRO prints an asterisk, at which time another command line may be entered.

| <u>Error Message</u> | <u>Explanation</u>   |
|----------------------|--|
| ?C-CHAIN-ONLY-CUSP?  | An attempt was made either to "R CREF" or to "START" a copy of CREF which was in memory due to a previous MACRO run which had chained to CREF but was aborted from the console terminal (via CTRL C). CREF can only be "chained" to; it cannot be invoked without MACRO. |
| ?C-CRF FILE ERROR?   | An output error occurred while accessing "DK:CREF.TMP", the temporary file passed from MACRO to CREF.  |
| ?C-DEVICE?           | An invalid device was specified to CREF by MACRO (system error).   |
| ?C-LST FILE ERROR?   | An output error occurred while attempting to write the cross-reference table to the listing file.  |



## CHAPTER 6

### LINKER

#### 6.1 INTRODUCTION

The RT-11 Linker converts object modules produced by either one of the RT-11 assemblers or FORTRAN IV into a format suitable for loading and execution. This allows the user to separately assemble a main program and each of its subroutines without assigning an absolute load address at assembly time. The object modules of the main program and subroutines are processed by the Linker to:

1. Relocate each object module and assign absolute addresses
2. Link the modules by correlating global symbols defined in one module and referenced in another module
3. Create the initial control block for the linked program
4. Create an overlay structure if specified and include the necessary run-time overlay handlers and tables
5. Search user specified libraries to locate unresolved globals
6. Optionally produce a load map showing the layout of the load module

The RT-11 Linker requires two or three passes over the input modules. During the first pass it constructs the global symbol table, including all control section names and global symbols in the input modules. If library files are to be linked with input modules, an intermediate pass is needed to force the modules resolved from the library file into the root segment (that part of the program which is never overlaid). During the final pass, the Linker reads the object modules, performs most of the functions listed above, and produces a load module (LDA for use with the Absolute Loader, save image (SAV) for a Single-job system or for the background job of an F/B System, and relocatable (REL) format for the foreground job of an F/B System).

The Linker requires at least 8K of memory; any additional memory is used to extend the symbol table. Input is accepted from any binary device on the system; there must be at least one random access device (disk or DECTape) for save image or relocatable format output.



## Linker

### 6.2 CALLING AND USING THE LINKER

To call the Linker, type the command:

R LINK

and the RETURN key in response to the Keyboard monitor's dot. The Linker prints an asterisk and awaits a command string.

Type CTRL C to halt the Linker at any time and return control to the monitor. To restart the Linker, type R LINK or the REENTER command in response to the monitor's dot. The Linker outputs an extra line feed character when it is restarted with REENTER or after an error in the first command line. When the Linker is finished linking, control returns to the CSI automatically. An extra line feed character precedes the asterisk printed by the CSI.

#### 6.2.1 Command String

The first command string entered in response to the Linker's asterisk has the following format:

\*dev:binout,dev:mapout=dev:obj1,dev:obj2,.../s1/s2/s3

where:

|           |   |
|-----------|---|
| dev:      | is a random access device for the save image or REL format output file (binout) and any appropriate device in all other instances. If dev: is not specified, DK is assumed. If the output is to be LDA format (that is, the /L switch was used), the output file need not be on a random access device. |
| binout    | is the name to be assigned to the Linker's save image, LDA format, or REL format output file. This file is optional; if not specified, no binary output is produced. (Save image is the assumed output format unless the /L or /R switches are used.)   |
| mapout    | is the optional load map file.  |
| obj1,...  | are files of one or more object modules to be input to the Linker (these may be library files).   |
| /s1/s2/s3 | are switches as explained in Table 6-1 and Section 6.8.   |

If the /C switch is given, subsequent command lines may be entered as:

\*objm,objn,.../s1/s2

The /C switch is necessary only if the command string will not fit on one line or if the overlay structure is used. If an error occurs in a continued command line (e.g., ?FILE NOT FND?), only the line in error need be retyped.



## Linker

If an output file is not specified, the Linker assumes that the associated output is not desired. For example, if the load module and load map are not specified, only error messages (if any) are printed by the Linker.

The default values for each specification are:

|               | <u>Device</u>                               | <u>Filename</u> | <u>Extension</u>      |
|---------------|---|-----------------|-----------------------|
| Load Module   | DK:   | none            | SAV, REL(/R), LDA(/L) |
| Map Output    | Same as<br>load module                      | none            | MAP                   |
| Object Module | DK: or same<br>as previous<br>object module | none            | OBJ                   |

If a syntax error is made in a command string, an error message is printed. A new command string can then be typed following the asterisk.

If a nonexistent file is specified a fatal error occurs; control is returned to the command string interpreter, an asterisk is printed and a new command string may be entered.

### 6.2.2 Switches

The switches associated with the Linker are listed in Table 6-1. The letter representing each switch is always preceded by the slash character. Switches must appear on the line indicated if the command is continued on more than one line. They may be positioned anywhere on the line. (A more detailed explanation of each switch is provided in Section 6.8.)

Table 6-1  
Linker Switches

| Switch Name | Command Line | Meaning  |
|-------------|--------------|--|
| /A          | 1st          | Alphabetizes the entries in the load map.  |
| /B:n        | 1st          | Bottom address of program is indicated as n (illegal for foreground links).  |
| /C          | any          | Continues input specification on another command line. Used also with /O.  |
| /F          | 1st          | Instructs the Linker to use the default FORTRAN library, FORLIB.OBJ; note that FORLIB does not have to be specified in the command line. |
| /I          | 1st          | Includes the global symbols to be searched from the library.   |
| /L          | 1st          | Produces an output file in LDA format (illegal for foreground links).  |

(Continued on next page)



Table 6-1 (Cont.)  
Linker Switches

| Switch Name | Command Line    | Meaning   |
|-------------|-----------------|---|
| /M or /M:n  | 1st             | Stack address is to be specified at the terminal keyboard or via n.   |
| /O:n        | any but the 1st | Indicates that the program will be an overlay structure; n specifies the overlay region to which the module is assigned.  |
| /R          | 1st             | Produces output in REL format; only files in REL format will run in the foreground (REL format files may not be run under a Single-Job system)  |
| /S          | 1st             | Allows the maximum amount of space in memory to be available for the Linker's symbol table. (This switch should only be used when a particular link stream causes a symbol table overflow.) |
| /T or /T:n  | 1st             | Transfer address is to be specified at terminal keyboard or via n.  |

## 6.3 ABSOLUTE AND RELOCATABLE PROGRAM SECTIONS

A program produced by one of the RT-11 assemblers or FORTRAN IV can consist of an absolute program section, declared by the .ASECT assembler directive, and relocatable program sections declared by the .CSECT assembler directive. A .CSECT directive is assumed at the beginning of the source program. The instructions and data in relocatable sections are normally assigned locations beginning at 1000(octal) or 0 for a foreground link. The assignment of addresses can be influenced by command string switches and the size of the absolute section (.ASECT, if present). Each control section is assigned a memory address; the Linker then appropriately modifies all instructions and/or data as necessary to account for the relocation of the control sections.

## NOTE

Foreground programs cannot use .ASECTs beyond 1000 (octal); as a general practice, they should be avoided under a Foreground/Background system.

The RT-11 Linker handles the absolute section as well as the named and unnamed control sections. The unnamed control section is internal to each object module. That is, every object module can have an unnamed control section but the Linker treats each control section independently. Each is assigned an absolute address such that it occupies an exclusive area of memory. Named control sections, on the other hand, are treated globally; if different object modules have control sections with the same name, they are all assigned the same absolute load address and the size of the area reserved for loading of the section is the size of the largest. Thus, named control sections allow for the sharing of data and/or instructions among object modules. This is the same as the handling and function of COMMON in FORTRAN IV. The names assigned to control sections are global and can be referenced as any other global symbol.



## Linker

### 6.4 GLOBAL SYMBOLS

Global symbols provide the link, or communication, between object modules. Global symbols are created with the `.GLOBL` assembler directive (see Chapter 5). If the global symbol is defined in an object module (as a label or by direct assignment), it is called an entry symbol and other object modules can reference it. If the global symbol is not defined in the object module, it is an external symbol and is assumed to be defined (as an entry symbol) in some other object module.

As the Linker reads the object modules it keeps track of all global symbol definitions and references. It then modifies the instructions and/or data which reference the global symbols. Undefined globals are printed on the console terminal after pass 1 (or pass 2 if a library file is also linked).

### 6.5 INPUT AND OUTPUT

Linker input and output is in the form of modules; one or more input modules (object files produced by either assembler or FORTRAN IV) are used to produce a single output (load) module.

#### 6.5.1 Object Modules

Object files, consisting of one or more object modules, are the input to the Linker (the Linker ignores files which are not object modules). Object modules are created by the RT-11 assemblers or FORTRAN IV. The Linker reads each object module at least twice (three times if library files are linked). During the first pass each object module is read to construct a global symbol table and to assign absolute values to the control section names and global symbols. If library files are linked, a second pass is needed to resolve the undefined globals from the library files and force their associated object modules into the root; on the final pass, the Linker reads the object modules, links and relocates the modules and outputs the load module.

#### 6.5.2 Load Module

The primary output of the Linker is a load module which may be loaded and run under RT-11. The load module is output as a save image file (SAV) for use under a Single-Job system or the background job. Under an F/B System, the `/R` switch must be used to produce a REL (reloctable) format foreground load module if the job is to run in the foreground. An absolute load module (LDA) is produced if the module is to be loaded by the Absolute Loader.

The load module for a save image file is arranged as follows:

|              |                                   |
|--------------|-----------------------------------|
| Root Segment | Overlay<br>Segments<br>(optional) |
|--------------|-----------------------------------|

For a REL image file, the load modules are arranged as:



## Linker

| Root Segment | Overlay Segments (optional) | Resident REL Blocks | Overlay REL Blocks (optional) |
|--------------|-----------------------------|---------------------|-------------------------------|
|--------------|-----------------------------|---------------------|-------------------------------|

The first 256-word block of the root segment (main program) contains the memory usage map and the locations used by the Linker to pass program control parameters. The memory usage map outlines the blocks of memory used by the load module and is located in locations 360 to 377.

The control parameters are located in locations 40-50 and contain the following information when the module is loaded:

| <u>Address</u> | <u>Information</u>                           |
|----------------|--|
| 40:            | Start Address of program                     |
| 42:            | Initial setting of R6 (stack pointer)        |
| 44:            | Job Status Word                              |
| 46:            | USR Swap Address (0 implies normal location) |
| 50:            | Highest Memory Address in user's program     |

For a foreground link the following additional parameters contain information:

|        |  |
|--------|--|
| 34,36: | TRAP Vector  |
| 52:    | Size of Resident (words)                               |
| 54:    | Sum of the Resident and largest Overlay Region (words) |
| 56:    | F/B Identification                                     |
| 60:    | Address of Resident REL Block                          |

Memory locations 0-476 (comprising the interrupt vectors and system communication area) may be assigned initial values by using an .ASECT assembler statement and will appear in block 0 of the load module, but there are restrictions on the use of .ASECTs in this region. The Linker does not permit an .ASECT of location 54 or of locations 360-377 (the memory usage map is passed in those locations). In addition, for foreground links, an .ASECT of words 52-60 is not permitted (additional parameters are passed to FRUN in those locations).

Any location which is not restricted may be set with an .ASECT, but caution should be used in changing the system communication area. Restricted areas, such as the region 360-377, must be initialized by the program itself. There are no restrictions on .ASECTs if the output format is LDA.

Locations in the region 0-476 which are initialized by an .ASECT in a program may never be loaded when the program is executed. There are two reasons for this. For background tasks (or the Single-Job system) the R, RUN, and GET commands will not load an address protected by the monitor's memory protection map. The addresses normally protected include such important areas as the system device and console device vectors, but protection may be extended dynamically (e.g., by a foreground task issuing a .PROTECT call). For foreground tasks, the FRUN command will load only locations 34-50 (34 is the TRAP instruction vector) and all other .ASECTs are ignored. The procedure for loading these locations is to do so at run-time using MOV instructions.



## Linker

### 6.5.3 Load Map

If requested, a load map is produced following the completion of the initial pass(es) of the Linker. This map, shown in Figure 6-1, diagrams the layout of memory for the load module.

Each.CSECT included in the linking process is listed in the load map. The entry for a.CSECT includes the name and low address of the section and its size (in bytes). The remaining columns contain the entry points (or globals) found in the section and their addresses.

The map begins with the name of the load module and the date of creation. The modules located in the root segment of the load module are listed next, followed by those modules which were assigned to overlays in order by their region number (see Section 6.6). Any undefined global symbols are then listed. The map ends with the transfer address (start address) and high limit of relocatable code.

#### NOTE

The load map will not reflect the absolute addresses for a REL file created to be run as a foreground job; the base relocation address specified at FRUN time must be added to obtain the absolute addresses.

For example, assume the FRUN command is used to run the program RELTST:

```
.FRUN RELTST/F
LOADED AT 137150
```

When linked, the following load map is produced:

| RT=11 LINK |        | X03-16 |        | LOAD MAP  |        |        |        |        |
|------------|--------|--------|--------|-----------|--------|--------|--------|--------|
| RELTST.REL |        |        |        | 03-SEP-74 |        |        |        |        |
| SECTION    | ADDR   | SIZE   | ENTRY  | ADDR      | ENTRY  | ADDR   | ENTRY  | ADDR   |
| , ABS.     | 000000 | 000350 | STKSIZ | 000012    | SFVEC  | 000320 | SCRPOS | 001750 |
| TSTVT1     | 000350 | 000326 |        |           |        |        |        |        |
| SGTB       | 000676 | 001116 | SDBINT | 000676    | SSTOPF | 000710 | SNR    | 000752 |
|            |        |        | STACKP | 000764    | STATBF | 001106 | SDPC   | 001142 |
|            |        |        | SLINKF | 001146    | SLCDIS | 001150 | SLCNT  | 001152 |
|            |        |        | SDBVEC | 001154    | STACKE | 001202 | SDFILE | 001226 |
|            |        |        | SY8    | 001244    | SBLANK | 001246 | SLINK  | 001252 |
|            |        |        | S8YPAS | 001256    | SCTRAK | 001260 | SLSRA  | 001276 |
|            |        |        | SCUSER | 001302    | SNULL  | 001316 | SXT    | 001326 |
|            |        |        | SYT    | 001330    | SLPINT | 001404 | SLPBUF | 001434 |
|            |        |        | SNRBUF | 001446    | STRAKC | 001562 | SXSTOR | 001572 |
|            |        |        | SYSTOR | 001574    | S8OINT | 001670 | SVSTIN | 001676 |
|            |        |        | SVSTP  | 001722    | SPDV1  | 001756 | SPEXIT | 001764 |



#### Linker

|                       |                |               |               |  |
|-----------------------|----------------|---------------|---------------|--|
| OVERLAY REGION 000001 | SEGMENT 000001 |               |               |  |
| SGT1 002016 000210    | SVINIT 002016  | SVFDEL 002100 | SVSTOP 002114 |  |
|                       | SVNSRT 002130  | SVRMOV 002174 | SVSTRT 002214 |  |
| SGT2 002226 000102    | SVBLNK 002226  | SVRSTR 002256 | SVSRCH 002300 |  |
| SGT3 002330 000130    | SVTRAK 002330  | SVLPEN 002410 | SVSTPM 002420 |  |
|                       | SNOSYN 002430  | SSYNC 002440  | SNAME 002450  |  |
| OVERLAY REGION 000001 | SEGMENT 000002 |               |               |  |
| SGT4 002016 000602    | SVRTLK 002016  | SVUNLK 002300 | SVLSET 002404 |  |
|                       | SVSCRL 002512  |               |               |  |

TRANSFER ADDRESS = 000350  
HIGH LIMIT = 002620

To determine the address of TSTVT1, 137150 must be added to 000350; thus 137520 is the absolute address of TSTVT1. The transfer address is 137150 plus 350, or 137520.

#### 6.5.4 Library Files

The RT-11 Linker has the capability of automatically searching libraries. Libraries are composed of library files--specially formatted files produced by the Librarian program (Chapter 7) which contain one or more object modules. The object modules provide routines and functions to aid the user in meeting specific programming needs. (For example, FORTRAN has a special library containing all necessary computational functions--TAN, ATAN, etc.) By using the Librarian, libraries can be created and updated so that routines which are used more than once, or routines which are used by more than one program, may be easily accessed. Selected modules from the appropriate library file are linked as needed with the user program to produce one load module. Libraries are further described in Section 6.7 and in Chapter 7.

#### NOTE

Library files that have been combined under PIP are illegal as input to both the Linker and the Librarian.



# Linker

| RT-11 LINK |                 | V03-01 | LOAD MAP  |        |        |        |         |        |
|------------|-----------------|--------|-----------|--------|--------|--------|---------|--------|
| SQRT .SAV  |                 |        | 19-SEP-74 |        |        |        |         |        |
| SECTION    | ADDR            | SIZE   | ENTRY     | ADDR   | ENTRY  | ADDR   | ENTRY   | ADDR   |
| , ABS.     | 000000          | 001000 | SUSRSW    | 000000 | SV005A | 000001 | SNLCHN  | 000006 |
|            |                 |        | SLRECL    | 000210 | STRACE | 004737 |         |        |
|            | 001000          | 000220 |           |        |        |        |         |        |
|            | 001220          | 001364 | SOTI      | 001246 |        |        |         |        |
|            | 002604          | 002300 | OCIS      | 002604 | ICIS   | 002612 | SGET    | 002772 |
|            |                 |        | RCIS      | 003006 | OCOS   | 003712 | ICOS    | 003720 |
|            |                 |        | GCOS      | 004144 | FCOS   | 004152 | ECOS    | 004156 |
|            |                 |        | DCOS      | 004164 |        |        |         |        |
|            | 005104          | 000160 | ISNS      | 005104 | SISNTR | 005110 | LSNS    | 005124 |
|            |                 |        | SLSNTR    | 005130 |        |        |         |        |
|            | 005264          | 000102 | MOISSS    | 005264 | MOLSSS | 005264 | MOISSM  | 005270 |
|            |                 |        | MOISSA    | 005274 | MOISIS | 005300 | MOLSIIS | 005300 |
|            |                 |        | RELS      | 005300 | MOISIM | 005304 | MOISIA  | 005310 |
|            |                 |        | MOISMS    | 005314 | MOISMM | 005320 | MOISMA  | 005324 |
|            |                 |        | MOISOS    | 005330 | MOISOM | 005334 | MOISOA  | 005340 |
|            |                 |        | MOISIS    | 005344 | MOISIM | 005352 | MOISIA  | 005360 |
|            | 005366          | 000020 | IFRS      | 005366 | IFWS   | 005400 |         |        |
|            | 005406          | 000046 | EOLS      | 005406 |        |        |         |        |
|            | 005454          | 000062 | TVLS      | 005454 | TVFS   | 005462 | TVDS    | 005470 |
|            |                 |        | TVQS      | 005476 | TVPS   | 005504 | TVIS    | 005512 |
|            | 005536          | 000036 | CAIS      | 005536 | CALS   | 005544 |         |        |
|            | 005574          | 000174 | SQRT      | 005574 |        |        |         |        |
|            | 005770          | 000026 | MOFSRS    | 005770 | MOFSRM | 005776 | MOFSRA  | 006006 |
|            |                 |        | MOFSRP    | 006012 |        |        |         |        |
|            | 006016          | 000044 | NMISIM    | 006016 | NMISII | 006026 | BLES    | 006034 |
|            |                 |        | BEQS      | 006036 | BGTS   | 006044 | BGES    | 006046 |
|            |                 |        | BRAS      | 006050 | BNES   | 006054 | BLTS    | 006056 |
|            | 006062          | 000072 | FOOS      | 006062 | EXIT   | 006074 | STPS    | 006074 |
|            | 006154          | 000002 | SAOTS     | 006154 |        |        |         |        |
|            | SERRTB<br>SERRS | 006156 | 000100    |        |        |        |         |        |
| 006256     |                 | 002637 |           |        |        |        |         |        |
| 011116     |                 | 001534 | SFIO      | 011600 |        |        |         |        |
| 012652     |                 | 000202 | SFMTDR    | 012652 | SFMTDW | 012702 | SINITI  | 012750 |
| 013054     |                 | 000416 | SCLOSE    | 013054 |        |        |         |        |
| 013472     |                 | 000106 | LCIS      | 013472 | LCOS   | 013540 |         |        |
| 013600     |                 | 000302 | SGETRE    | 013600 | STTYIN | 013722 |         |        |
| 014102     |                 | 000262 | SPUTRE    | 014102 |        |        |         |        |
| 014364     |                 | 000106 | SFCHNL    | 014364 |        |        |         |        |
| 014472     |                 | 000674 | SOPEN     | 014472 |        |        |         |        |
| 015366     |                 | 000110 | SDUMPL    | 015366 |        |        |         |        |
| 015476     |                 | 000414 | SPUTBL    | 015476 | SGETBL | 015676 | SEOFIL  | 016046 |
| 016112     |                 | 000042 | SWAIT     | 016112 |        |        |         |        |

TRANSFER ADDRESS = 001000  
HIGH LIMIT = 016154

Figure 6-1  
Linker Load Map for Background Job



## Linker

### 6.6 USING OVERLAYS

The RT-11 program overlay facility enables the user to have virtually unlimited memory space for an assembly language or FORTRAN program. A program using the overlay facility can be much larger than would normally fit in the available memory space, since portions of the program (called overlay segments) reside on a backup storage device (disk or DECTape).

The RT-11 overlay scheme is a strict multi-region arrangement; it is not tree-structured. Figure 6-2 diagrams this scheme. The overlay system which the user constructs from his completed program is composed of a root segment, memory-resident overlay regions, and the overlay segments stored on the backup storage device. The root segment is a required part of every overlay program and contains all transfer addresses; it must therefore never be overlaid. An overlay region corresponds to a run-time area of memory that is shared by two or more subroutines (called co-resident subroutines); there is a distinct memory area for each overlay region. Overlay segments are portions of the save image or REL format file from which the user's program is run; these are brought into memory as needed.

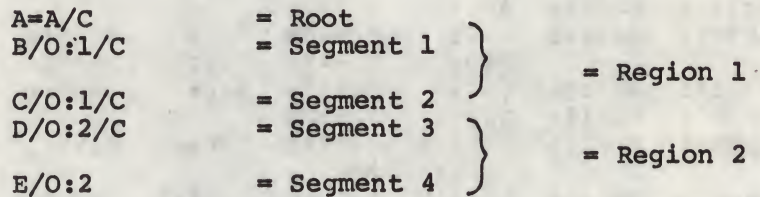


Figure 6-2  
Overlay Scheme

Overlay regions are specified to the Linker via the /O switch as described in Section 6.8.8. The size of the overlay region is calculated by the Linker to be the size of the largest group of subroutines that can occupy the region at one time. The Linker creates the overlay regions and edits the program to produce the desired overlays at run-time.

Figure 6-3 shows a diagram of memory for a program which has an overlay structure and Figure 6-4 is a listing of the run-time overlay handler.



# Linker

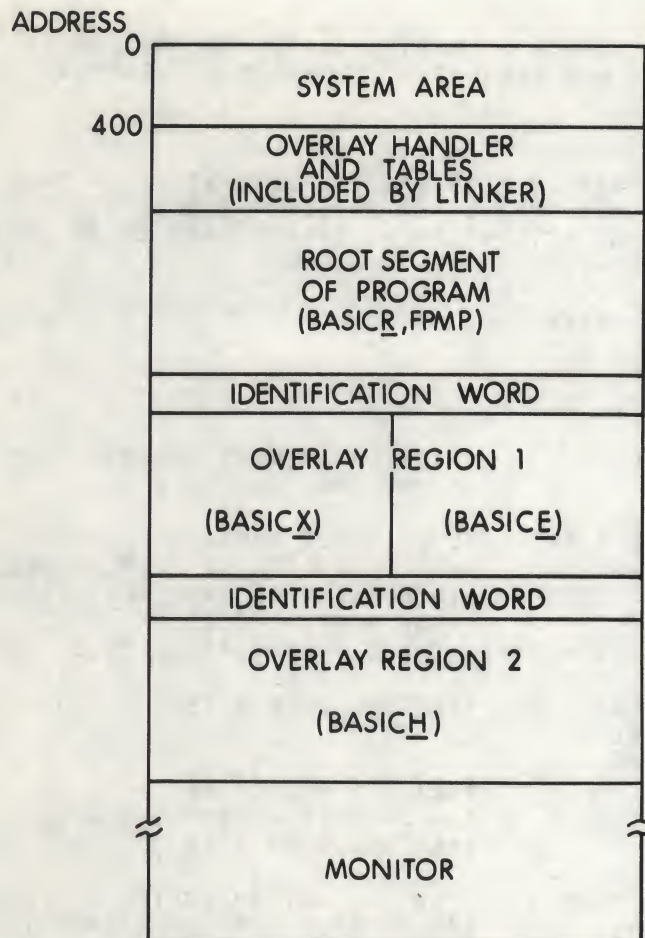


Figure 6-3  
Memory Diagram Showing BASIC Link With Overlay Regions

## .SBTTL THE RUN-TIME OVERLAY HANDLER

; THE FOLLOWING CODE IS INCLUDED IN THE USER'S PROGRAM BY THE  
 ; LINKER WHENEVER OVERLAYS ARE REQUESTED BY THE USER.  
 ; 56.8 MICROSECONDS (APPROX) IS ADDED TO EACH REFERENCE OF  
 ; A RESIDENT OVERLAY SEGMENT.

; THE RUN-TIME OVERLAY HANDLER IS CALLED BY A DUMMY  
 ; SUBROUTINE OF THE FOLLOWING FORM:

```

;           JSR      RS,SOVRH           ;CALL TO COMMON CODE
;           .WORD    <OVERLAY #>       ;# OF DESIRED SEGMENT
;           .WORD    <ENTRY ADDR>      ;ACTUAL CORE ADDR
  
```

; ONE DUMMY ROUTINE OF THE ABOVE FORM IS STORED IN THE RESIDENT  
 ; PORTION OF THE USER'S PROGRAM FOR EACH ENTRY POINT TO  
 ; AN OVERLAY SEGMENT. ALL REFERENCES TO THE ENTRY POINT ARE  
 ; MODIFIED BY THE LINKER TO INSTEAD BE REFERENCES TO THE APPRO-  
 ; PRIATE DUMMY ROUTINE. EACH OVERLAY SEGMENT IS CALLED INTO  
 ; CORE AS A UNIT AND MUST BE CONTIGUOUS IN CORE. AN OVERLAY



# Linker

; SEGMENT MAY HAVE ANY NUMBER OF ENTRY POINTS, TO THE LIMITS  
 ; OF CORE MEMORY. ONLY ONE SEGMENT AT A TIME MAY OCCUPY AN  
 ; OVERLAY REGION.

; RESTRICTIONS:  
 ; SINCE REFERENCES TO OVERLAY SEGMENTS ARE AUTOMATICALLY TRANS-  
 ; LATED BY THE LINKER INTO REFERENCES TO DUMMY SUBROUTINES,  
 ; THE PROGRAMMER MUST NOT ATTEMPT TO REFERENCE DATA IN AN OVER-  
 ; LAY BY USING GLOBAL SYMBOLS.

```

SOVRTAB*1000+SOVRHE=SOVRH
SOVRH:  MOV    R0,=(SP)
        MOV    R1,=(SP)
        MOV    R2,=(SP)
SOVRHB:
;      MOV    (R5)+,R0      ;PICK UP OVERLAY NUMBER
;      BR     $FIRST      ;FIRST CALL ONLY * * *
;      MOV    R0,R1
SOVRHA: ADD    #SOVRTAB-6,R1 ;CALC TABLE ADDR
        MOV    (R1)+,R2      ;GET CORE ADDR OF OVERLAY REGION
        CMP    R0,R2        ;IS OVERLAY ALREADY RESIDENT?
        BEQ    $ENTER      ;YES, BRANCH TO IT
        .READW 17,R2,(R1)+,(R1)+ ;READ FROM OVERLAY FILE
        BCS    $ERR
$ENTER: MOV    (SP)+,R2      ;RESTORE USER'S REGS
        MOV    (SP)+,R1
        MOV    (SP)+,R0
        MOV    @R5,R3      ;GET ENTRY ADDRESS
        RTS     R5         ;ENTER OVERLAY ROUTINE AND
                           ;RESTORE USER'S R5

$FIRST: MOV    #12500,$QVRHB ;RESTORE SWITCH INSTR
        MOV    (PC)+,R1     ;START ADDR FOR CLEAR OPERATION
SHROOT: .WORD   0           ;HIGH ADDR OF ROOT SEGMENT
        MOV    (PC)+,R2     ;COUNT
SHOVLY: .WORD   0           ;HIGH LIMIT OF OVERLAYS
151     CLR     (R1)+       ;CLEAR ALL OVERLAY REGIONS
        CMP    R1,R2
        BLO    15
        BR     $QVRHB      ;AND RETURN TO CALL IN PROGRESS
$ERR:   EMT     376         ;GENERATE ALWAYS FATAL ERROR
        .BYTE  0,373       ;AND DISREGARD SOFT ERROR
SOVRHE:
  
```

; OVERLAY SEGMENT TABLE FOLLOWS:  
 ; SOVRTAB: .WORD <CORE ADDR>,<RELATIVE BLK>,<WORD COUNT>  
 ; THREE WORDS PER ENTRY, ONE ENTRY PER OVERLAY SEGMENT.

; ALSO, THERE IS ONE WORD PREFIXED TO EACH OVERLAY REGION  
 ; THAT IDENTIFIES THE SEGMENT CURRENTLY RESIDENT IN THAT REGION.  
 ; THIS WORD IS AN INDEX INTO THE SOVRTAB TABLE.

Figure 6-4  
 The Run-Time Overlay Handler



## Linker

There is no special code or function call needed to use overlays but the following rules must be observed when referencing parts of the user program which might be overlaid.

1. Calls or branches to overlay segments must be made directly to entry points in the segment. Entry points are locations tagged with a global symbol (refer to Chapter 5, Section 5.5.10). For example, if ENTER is a global tag in an overlay segment:

|             |               |
|-------------|---------------|
| JMP ENTER   | is legal, but |
| JMP ENTER+6 | is illegal.   |

2. Entries in overlay segments can be used only for transfer of control and not for referencing data within an overlay section (e.g., MOV ENTER,R4 is illegal if ENTER is in an overlay segment, but MOV #ENTER,R7 is legal because it is used for transfer of control). A violation of this rule cannot be detected by the assembler or Linker so no error is issued; however, it can cause the program to use incorrect data.
3. When calls are made to overlays, the entire return path must be in memory. This will happen if these rules are followed:

Calls (with expected return) may be made from an overlay segment only to entries in the same segment, the root segment, or an overlay segment with a greater region number.

Calls to entries in the same region as the call must be entirely within the same segment, not another segment in the same region.

Jumps (with no expected return) can be made from an overlay segment to any entry in the program. However, jumps should not reference an overlay region whose number is lower than the region from which the last unreturned call was made (e.g., if a call was made from region 3, then no jumps should reference regions 1, 2 or 3 until the call has returned).

Subroutines in the root segment may be called from overlay segments; in turn, they may call entries from the same overlay segment which called them, or from the root segment, or from another overlay segment with a greater region number. Such subroutines are considered part of the overlay segment which called them.

4. A .CSECT name cannot be used to pass control to an overlay. It will not cause the appropriate segment to be loaded into memory (e.g., JSR PC,OVSEC is illegal if OVSEC is used as a .CSECT name in an overlay). As stated in 1 above, a global symbol must be used to pass control from one segment to the next.
5. Channel.17(octal) cannot be used by the user program because overlays are read on that channel.
6. Object modules acquired from a library file cannot be placed into overlays.



## Linker

7. Library files may not be specified on the same command line as an overlay.
8. Overlay regions must be specified in ascending order and are read-only. Unlike USR swapping, an overlay segment does not save the segment it is overlaying. Any tables, variables, or instructions that are modified within a given overlay segment are re-initialized to their original values in the SAV or REL file if that segment has been overlaid by another segment. Any variables or tables whose values must be maintained across overlays should be placed in the root segment.
9. .ASECTs of any size in an overlay foreground link are illegal; the error message ?ILL ASECT? is printed and the line is aborted.

The following information should be noted when writing FORTRAN overlays.

1. When dividing a FORTRAN program into a root segment and overlay regions (and subsequently dividing each overlay region into overlay segments), routine placement should be carefully considered. The user should always remember that it is illegal to call a routine located in a different overlay segment in the same overlay region, or an overlay region with a lower numeric value (as specified by the Linker overlay switch, /O:n) from the calling routine. The user should divide each overlay region into overlay segments which never need to be resident simultaneously (i.e., if segments A and B are assigned to region X, they cannot call each other because they occupy the same locations in memory).
2. The FORTRAN main program unit must be placed in the root segment.
3. In an overlay environment, subroutine calls and function subprogram references may refer only to one of the following:
  - A FORTRAN library routine (e.g., ASSIGN, DCOS)
  - A FORTRAN or assembly language routine contained in the root segment
  - A FORTRAN or assembly language routine contained in the same overlay segment as the calling routine
  - A FORTRAN or assembly language routine contained in a segment whose region number is greater than that of the calling routine
4. In an overlay environment, COMMON blocks must be placed so that they are resident when referenced. Blank COMMON is always resident since it is always placed in the root segment. All named COMMON must be placed either in the root segment, or into the segment whose region number is lowest of all segments which reference the COMMON block. A named COMMON block cannot be referenced by two segments in the same region unless the COMMON block appears in a segment of a lower region number. The Linker automatically places a COMMON block into the root segment if it is referenced by the FORTRAN main program or by a subprogram that is located in



## Linker

the root segment. Otherwise the Linker places a COMMON block in the first segment encountered in the Linker command string that references that COMMON block.

5. All COMMON blocks which are initialized (by use of DATA statements) must be so initialized in the segment in which they are placed.

Refer to the RT-11 FORTRAN COMPILER AND OBJECT TIME SYSTEM USER'S MANUAL (DEC-11-LRFP-A-D) for more details.

The .ASECT never takes part in overlaying in any way (i.e., if part of an .ASECT is destroyed by overlay operations, it is not restored by the overlay handler).

The aforementioned sets of rules apply only to communications among the various modules that make up a program. Internally, each module must only observe standard programming rules for the PDP-11 (as described in the PDP-11 PROCESSOR HANDBOOK and in Chapter 5).

It should be noted that the condition codes set by a user program are not preserved across overlay segment boundaries.

The Linker provides overlay services by including a small resident overlay handler (Figure 6-4) in the same file with the user program to be used at program run-time. This overlay handler plus some tables are inserted into the user's program beginning at the bottom address computed by the Linker. The Linker moves the user's program up in memory by an appropriate amount to make room for the overlay handler and tables, if necessary.

### 6.7 USING LIBRARIES

Libraries are specified in a command string in the same fashion as normal modules; they may be included anywhere in the command string, with the exception of overlay lines. If a global symbol is undefined at the time the library is encountered in the input stream and a module is included in the library which includes that global definition, that module is pulled from the library and linked into the load image. Only the modules needed to resolve references are pulled from the library; unused modules are not linked.

#### NOTE

Modules in one library may call modules from another library; however, the libraries must appear in the command string in the order in which they are called. For example, assume module X in library ALIB calls SQRT from the FORTRAN library. To correctly resolve all globals, the order of ALIB and the FORTRAN library should appear in the command line as:

\*Z=B,ALIB/F  
or \*Z=B,ALIB,FORLIB



## Linker

Module B is the root. It calls X from ALIB and brings X into the root. X in turn calls SQRT which is brought from FORLIB into the root.

FORTTRAN libraries cannot precede their root segment in a command line as this creates a bad transfer address. For example:

```
*X=ROOT/F
*X=ROOT,FORLIB
```

are legal, but:

```
*X=FORLIB,ROOT
```

is not. Unpredictable results will occur.

### 6.7.1 User Library Searches

Object modules from the named user libraries built by the Librarian are relocated selectively and linked by the Linker. The RT-11 Linker searches a specified library file during the library pass as follows (refer to Figure 6-5 for a flowchart representation of this process):

1. If there are any undefined globals in the Linker's table when a library is encountered in the command string, proceed to step 2; otherwise skip this library (go to step 5).
2. Read the library directory.
3. If any of the undefined globals can be defined by a module in this library, include the relevant module into the linked output file; otherwise, go to step 5.
4. If any undefined globals remain in the Linker's table and they have not been looked for in the library, return to step 2; otherwise go to step 5.
5. Close the library file.
6. Go to the next element in the command string.

This search method allows modules to appear in any order in the library. Any number of libraries may be specified in a link, and they may be positioned anywhere, with the exception of overlay segments and the restrictions noted in Section 6.7.



## Linker

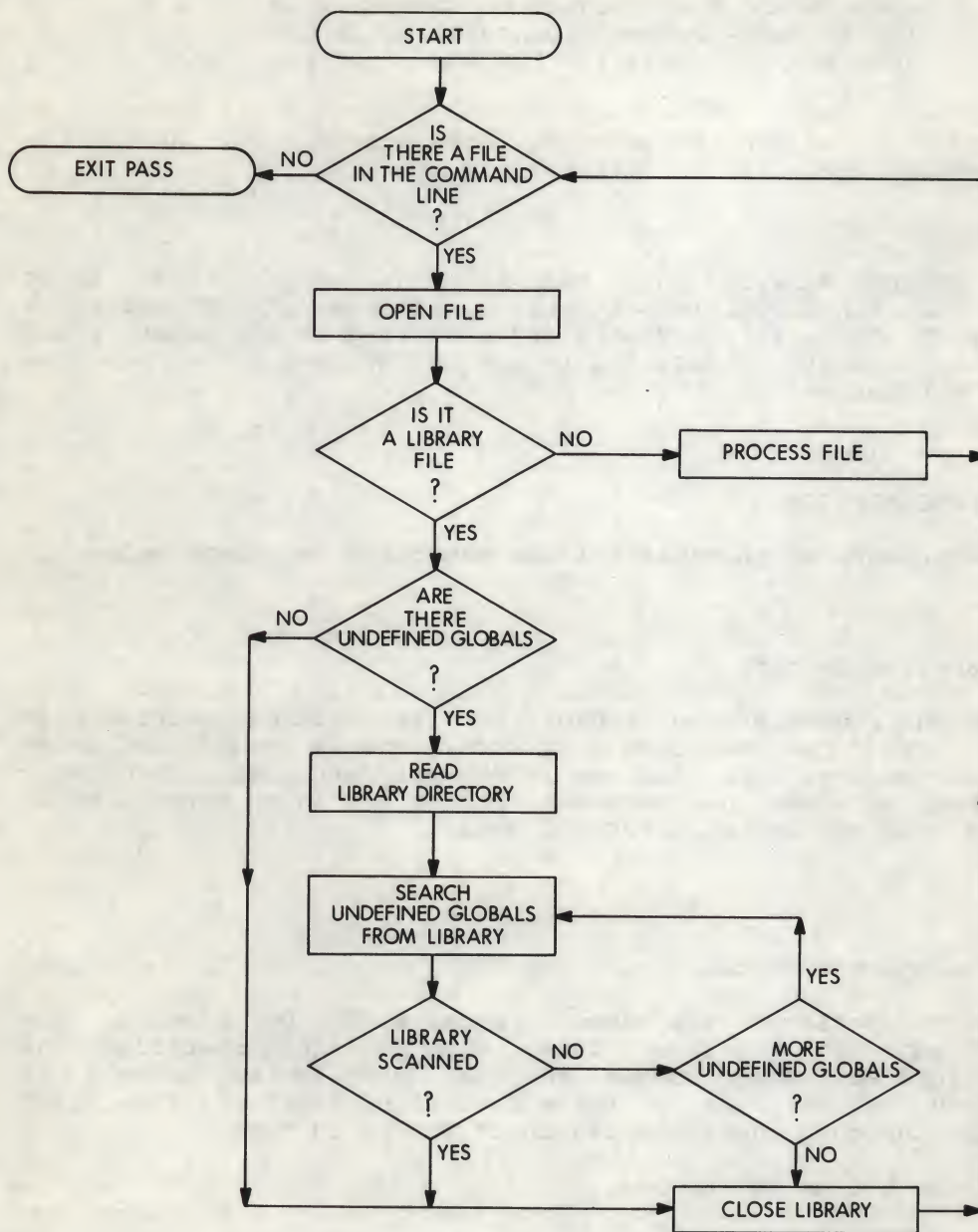


Figure 6-5  
Library Searches

### NOTE

For faster Linker performance, the user should specify all object files before library files, and all user library files before the system library files. For example:



## Linker

```
*A=A,B,USELIB/F
```

where A and B are object modules, USELIB is a user-created library file, and /F denotes the default FORTRAN library, FORLIB.

Libraries are input to the Linker as any other input file. Assume the following command string to the Linker:

```
*TASK01.SAV,LP:=MAIN.OBJ,MEASUR.OBJ
```

This causes program MAIN.OBJ to be read from DK: as the first input file. Any undefined symbols generated by program MAIN.OBJ should be satisfied by the library file MEASUR.OBJ specified in the second input file. The load module, TASK01.SAV is put on DK: and a load map goes to the line printer.

### 6.8 SWITCH DESCRIPTION

The switches summarized in Table 6-1 are described in detail below.

#### 6.8.1 Alphabetize Switch

The /A switch requests the Linker to list linked modules in alphabetical order as follows: .CSECTs, module names, and entry points within modules. The load map is normally arranged in order by module address as shown in Figure 6-1. Figure 6-6 is an example of an alphabetized load map for a background job.

#### 6.8.2 Bottom Address Switch

The /B switch specifies the lowest address to be used by the relocatable code in the load module. When /B is not specified, the Linker positions the load module so that the lowest address is location 1000 (octal), or 0 for a foreground link. If the .ASECT length is greater than 1000, the length of .ASECT is used.

The form of the bottom switch is:

```
/B:n
```

n is a six-digit unsigned octal number which defines the bottom address of the program being linked. An error message results if n is not specified as part of the /B command.

If more than one /B switch is specified during the creation of a load module, the first /B switch specification is used.

#### NOTE

The bottom value must be an unsigned even octal number. If the value is odd, an error message is generated.



# Linker

RT-11 LINK V03-01 LOAD MAP  
SQRT .SAV 19-SEP-74

| SECTION | ADDR   | SIZE   | ENTRY  | ADDR   | ENTRY  | ADDR   | ENTRY  | ADDR   |
|---------|--------|--------|--------|--------|--------|--------|--------|--------|
| . ABS.  | 000000 | 001000 | SLRECL | 000210 | SNLCHN | 000006 | SUSRSW | 000000 |
|         |        |        | STRACE | 004737 | SV005A | 000001 |        |        |
|         | 001000 | 000220 |        |        |        |        |        |        |
|         | 001220 | 001364 | SOTI   | 001246 |        |        |        |        |
|         | 002604 | 002300 | DCOS   | 004164 | ECOS   | 004156 | FCOS   | 004152 |
|         |        |        | GCOS   | 004144 | ICIS   | 002612 | ICOS   | 003720 |
|         |        |        | QCIS   | 002604 | OCOS   | 003712 | RCIS   | 003006 |
|         |        |        | SGET   | 002772 |        |        |        |        |
|         | 005104 | 000160 | ISNS   | 005104 | LSNS   | 005124 | SISNTH | 005110 |
|         |        |        | SLSNTH | 005130 |        |        |        |        |
|         | 005264 | 000102 | MOISIA | 005310 | MOISIM | 005304 | MOISIS | 005300 |
|         |        |        | MOISMA | 005324 | MOISMM | 005320 | MOISMS | 005314 |
|         |        |        | MOISSA | 005274 | MOISSM | 005270 | MOISSS | 005264 |
|         |        |        | MOIS0A | 005340 | MOIS0M | 005334 | MOIS0S | 005330 |
|         |        |        | MOIS1A | 005360 | MOIS1M | 005352 | MOIS1S | 005344 |
|         |        |        | MOLSI5 | 005300 | MOLSSS | 005264 | RELS   | 005300 |
|         | 005366 | 000020 | IFRS   | 005366 | IFWS   | 005400 |        |        |
|         | 005406 | 000046 | EOLS   | 005406 |        |        |        |        |
|         | 005454 | 000062 | TVDS   | 005470 | TVFS   | 005462 | TVIS   | 005512 |
|         |        |        | TVLS   | 005454 | TVPS   | 005504 | TVQS   | 005476 |
|         | 005536 | 000036 | CAIS   | 005536 | CALS   | 005544 |        |        |
|         | 005574 | 000174 | SQRT   | 005574 |        |        |        |        |
|         | 005770 | 000026 | MOFSRA | 006006 | MOFSRM | 005776 | MOFSRP | 006012 |
|         |        |        | MOFSRS | 005770 |        |        |        |        |
|         | 006016 | 000044 | BEGS   | 006036 | BGES   | 006046 | BGTS   | 006044 |
|         |        |        | BLES   | 006034 | RLTS   | 006056 | BNES   | 006054 |
|         |        |        | BRAS   | 006050 | NMIS1I | 006026 | NMIS1M | 006016 |
|         | 006062 | 000072 | EXIT   | 006074 | FOOS   | 006062 | STPS   | 006074 |
|         | 006154 | 000002 | SAOTS  | 006154 |        |        |        |        |
| SERRTB  | 006156 | 000100 |        |        |        |        |        |        |
| SERRS   | 006256 | 002637 |        |        |        |        |        |        |
|         | 011116 | 001534 | SFIO   | 011600 |        |        |        |        |
|         | 012652 | 000202 | SFMTDR | 012652 | SFMTDW | 012702 | SINITI | 012750 |
|         | 013054 | 000416 | SCLOSE | 013054 |        |        |        |        |
|         | 013472 | 000106 | LCIS   | 013472 | LCOS   | 013540 |        |        |
|         | 013600 | 000302 | SGETRE | 013600 | STTYIN | 013722 |        |        |
|         | 014102 | 000262 | SPUTRE | 014102 |        |        |        |        |
|         | 014364 | 000106 | SFCHNL | 014364 |        |        |        |        |
|         | 014472 | 000674 | SOPEN  | 014472 |        |        |        |        |
|         | 015366 | 000110 | SDUMPL | 015366 |        |        |        |        |
|         | 015476 | 000414 | SEOFIL | 016046 | SGETBL | 015676 | SPUTBL | 015476 |
|         | 016112 | 000042 | SWAIT  | 016112 |        |        |        |        |

TRANSFER ADDRESS = 001000  
HIGH LIMIT = 016154

Figure 6-6  
Alphabetized Load Map for a Background Job



## Linker

/B is illegal with foreground links. (0 is assumed.)

Example:

```
*OUTPUT,LP:=INPUT/B:500
```

Causes the input file to be linked starting at location 500 (octal).

### 6.8.3 Continue Switch

The Continue switch (/C) is used to allow additional lines of command string input. The /C switch is typed at the end of the current line and may be repeated on subsequent command lines as often as necessary to specify all input modules for which memory is available. If memory is exceeded, an error message is output. A /C switch is not entered on the last line of input.

Example:

```
*OUTPUT,LP:=INPUT/C
*
```

Input is to be continued on the next line; the Linker prints an asterisk.

### 6.8.4 Default FORTRAN Library Switch

By indicating the /F switch in the command line, the FORTRAN library, FORLIB.OBJ on the default device (SY:), is linked with the other object modules specified; the user does not need to specify FORLIB. For example:

```
*FILE,LP:=AB.OBJ/F
```

The object module AB.OBJ and the FORTRAN library FORLIB are linked together to form a load module called FILE.SAV. (Note that the FORLIB default is SY:FORLIB.OBJ, not DK:FORLIB.OBJ.)

### 6.8.5 Include Switch

The /I switch allows subsequent entry at the keyboard of global symbols to be taken from any library and included in the linking process. When the /I switch is specified, the Linker prints:

LIBRARY SEARCH:

Reply with the list of global symbols to be included in the load module; type a carriage return to enter each symbol in the list. A carriage return alone terminates the list of symbols. This provides a method for forcing modules (which are not called by other modules) to be loaded from the library.

Example:

```
*OUTPUT,LP:=INPUT,XLIB/I
```

LIBRARY SEARCH:

Linker prints LIBRARY SEARCH:

```
A <CR>
GETSYM <CR>
CHAR <CR>
CHFLG <CR>
<CR>
```

User enters A, GETSYM, etc. which are to be included in the linking process. Each symbol is entered by typing a carriage return; the list is terminated by an additional carriage return.



## Linker

### 6.8.6 LDA Format Switch

The LDA format switch (/L) causes the output file to be in LDA format instead of save image format. The LDA format file can be output to any device including non-file structured devices such as paper tape or cassette and is useful for files which are to be loaded with the Absolute Loader. The default extension .LDA is assigned when the /L switch is used.

The /L switch cannot be used in conjunction with the overlay switch (/O) or in foreground links (/R).

Example:

```
*DK:OUT,LP:=IN,IN2/L
```

Links disk files IN and IN2; outputs an LDA format file OUT.LDA to the system device and a load map to the line printer.

### 6.8.7 Modify Stack Address

The stack address, location 42, is the address which contains the user's stack pointer. The /M switch allows terminal keyboard specification of the user's stack address.

The form of the switch is:

```
/M:n
```

n is an unsigned 6-digit octal number which defines the stack address. If n is not specified, the Linker prints the message:

```
STACK ADDRESS =
```

In this case, specify the global symbol whose value is the stack address. A number cannot be specified, and if a nonexistent symbol is specified, an error message is printed and the stack address is set to the system default (1000 for save files, 0 for REL).

Direct assignment (via .ASECT) of the stack address within the program takes precedence over assignment with the /M switch.

Example:

```
*OUTPUT=INPUT/M
```

```
STACK ADDRESS = BEG
```

### 6.8.8 Overlay Switch

The Overlay switch (/O) is used to segment the load module so that the entire program is not memory resident at one time (overlay feature). This allows programs larger than the available memory to be executed. The switch has the form:

```
/O:n
```



## Linker

where n is an unsigned octal number (up to six digits in length) specifying the overlay region to which the module is assigned. The /O switch must follow (on the same line) the specification of the object modules to which it applies, and only one overlay region can be specified on a command line. Overlay regions cannot be specified on the first command line as this is the root segment. Therefore, the /C continuation switch must be used.

Co-resident overlay routines (a group of subroutines which occupy the overlay region at the same time) are specified as follows:

```
*OBJA,OBJB,OBJC/O:n/C
*OBJD,OBJE/O:m/C
.
.
.
```

All modules mentioned until the next /O switch will be co-resident overlay routines. If at a later time the /O switch is given with the same value previously used (same overlay region), then the corresponding overlay area is opened for a new group of subroutines. The new group of subroutines will occupy the same locations in memory as the first group, but not at the same time. For example, if subroutines in object modules R and S are to be in memory together, but are never needed at the same time as T, then the following commands to the Linker make R and S occupy the same memory as T (but at different times):

```
*MAIN,LP:=ROOT/C
*R,S/O:1/C
*T/O:1
```

The above could also be written as:

```
*MAIN,LP:=ROOT/C
*R/O:1/C
*S/C
*T/O:1
```

This places S in a different overlay segment in the same region as R, where in the first example S and R are in the same segment.

Example:

```
*OUTPUT,LP:=INPUT/C           Establishes two overlay
*OBJA/O:1/C                     regions
*OBJB/O:2
```

Overlays must be specified in order of increasing region number. For example:

```
R LINK
*A=A/C
*B/O:1/C
*C/O:1/C
*D/O:1/C
*E,F/O:2/C
*G/O:3
```

The following overlay specification is illegal since the overlay regions are given in a random numerical order (an error message is printed in each case):



## Linker

```
,R LINK
*A=A/C
*D/O:2/C
*B/O:1/C
/O IGNORED
*C/O:1/C
/O IGNORED
*G/O:3/C
*H/O:3/C
*E,F/O:2
/O IGNORED
```

### 6.8.9 REL Format Switch

The REL format switch (/R) causes the output file to be in REL format for use as a foreground job under the F/B Monitor. REL format files must be used in a foreground job (they may not be used under a Single-Job system). The /R switch assigns the default extension .REL to the output file.

#### Example:

```
*DT2:FILEO,LP:=FILEI,NEXT/R
```

Disk files FILEI and NEXT are linked and output to DT2 as FILEO.REL; a load map is output to the line printer.

The /B and /L switches cannot be used with /R since a foreground REL job has no bottom address and is always relocated by FRUN. A ?BAD SWITCH? error message is generated if this is attempted.

### 6.8.10 Symbol Table Switch

Use of the symbol table switch in the command line instructs the Linker to allow the largest possible memory area for its symbol table at the expense of making the link process slower. With the /S switch, library directories are not made resident in memory, but are left on disk. For example:

```
*OUTF,LP:=INPUT.OBJ,LIBR1.OBJ,LIBR2.OBJ/S
```

The directories of the library files LIBR1 and LIBR2 are not brought into memory, resulting in more room in the symbol table but longer link time.

If the /S switch is not used and the memory available to the Linker is 10K or larger, the library directory is brought into memory (providing there is room); the directory is kept there until the library has been completely processed, thus reducing the size of the Linker's symbol table. If there is not enough room in memory for the directory (as is the case in an 8K system), the Linker determines this and leaves the directory on disk regardless of whether the /S switch was used or not.

The /S switch should be used only if an attempt to link a program failed because of symbol table overflow. Often, use of /S will allow the program to link.



## Linker

### 6.8.11 Transfer Address Switch

The transfer address is the address at which a program is to be started when executed via an R, RUN, or FRUN command. The Transfer Address switch (/T) allows terminal keyboard specification of the start address of the load module to be executed. This switch has the form:

/T:n

where n is a six-digit unsigned octal number which defines the transfer address. If n is not specified, the message:

TRANSFER ADDRESS =

is printed. In this case, specify the global symbol whose value is the transfer address of the load module, followed by a carriage return. A number cannot be specified in answer to this message. When a nonexistent symbol is specified, an error message is printed and the transfer address is set to 1 (so that the program cannot be executed).

If the transfer address specified is odd, the program does not start after loading and control returns to the monitor.

Direct assignment (.ASECT) of the transfer address within the program takes precedence over assignment with the /T switch. The transfer address assigned with a /T has precedence over that assigned with a .END assembly directive.

#### Example:

```
*PROG=PROG1,PROG2,ODT/T
TRANSFER ADDRESS =
0.ODT
```

The files PROG1.OBJ, PROG2.OBJ and ODT.OBJ are linked together and started at ODT's transfer address.

### 6.9 LINKER ERROR HANDLING AND MESSAGES

The following error messages can be output by the Linker. The messages enclosed in question marks are output to the terminal; the other messages are only warnings and are included in the load map. If a load map is not requested in the command string, all messages are output to the terminal.

| <u>Message</u>                          | <u>Meaning</u>   |
|---|--|
| ADDITIVE REF OF xxxxxx<br>AT SEG#yyyyyy | Rule 1 of overlay rules explained in Section 6.6 has been violated. xxxxxx represents the entry point; yyyyyy represents the segment number. |
| ?/B NO VALUE?                           | The /B switch requires an octal number as an argument.   |
| ?/B ODD VALUE?                          | The argument to the /B switch was not an unsigned even octal number.   |
| ?BAD GSD?                               | There is an error in the global symbol directory (GSD). The file is probably not a   |



## Linker

legal object module. This error message occurs on pass 1 of the Linker.

### BAD OVERLAY AT SEG# yyyyyy

Overlay tries to store text outside its region; check for a .ASECT in overlay. yyyyyy represents the segment number.

### ?BAD RLD?

There is an invalid relocation directory (RLD) command in the input file; the file is probably not a legal object module. The message occurs on pass 2 of the Linker.

### ?BAD SWITCH?

LINK did not recognize a switch specified on the first command line. On a subsequent command line, a bad switch causes this warning message but does not restart the Linker.

### ?BAD x SWITCH IGNORED?

LINK did not recognize a switch (x) specified in the command line. The switch is ignored and linking continues.

### BYTE RELOCATION ERROR AT xxxxxx

Linker attempted to relocate and link byte quantities but failed. xxxxxx represents the address at which the error occurred. Failure is defined as the high byte of the relocated value (or the linked value) not being all zero. In such a case, the value is truncated to 8 bits and the Linker continues processing (for save image and LDA files only; byte relocation is completely illegal for REL files).

### ?CORE?

There is not enough memory to accommodate the command or the resultant load module.

### ?ERROR ERROR?

An error occurred while the Linker was in the process of recovering from a previous system or user error.

### ?ERROR IN FETCH?

The device is not available.

### ?FILE NOT FND?

Input file was not found.

### ?FORLIB NOT FND?

The user indicated via the /F switch that the FORTRAN library, FORLIB, was to be linked with the other object modules in the command line, and the Linker could not find FORLIB.OBJ on the system device.

### ?HARD I/O ERROR?

A hardware error occurred; try the operation again.

### ?ILL ASECT?

The user has attempted to place an .ASECT above 1000 in a foreground link or to place an .ASECT into an overlay foreground link.



## Linker

|  |  |
|--|--|
| ?LDA FILE ERROR?                                     | There was a hardware problem with the device specified for LDA output or the device was full.  |
| ?/M ODD VAL?   | An odd value has been specified for the stack address. Control returns to the Linker and another command line may be indicated.  |
| ?MAP FILE ERROR?                                     | There was a hardware problem with the device specified for map output or the device is full.   |
| MULT DEF OF xxxxxx                                   | The symbol, xxxxxx, was defined more than once.  |
| ?NO INPUT?   | No input files were specified.   |
| ?OUTPUT FULL?  | The output device was full.  |
| /O IGNORED   | Overlays have been specified in the wrong order (see overlay restrictions); the overlay switch is ignored.   |
| ?REL FILE ERR?                                       | The Linker encountered a problem writing the REL file; try the operation again.  |
| ?SAV FILE ERR?                                       | The Linker encountered a problem writing the save image file; try the operation again.   |
| ?STACK ADDRESS UNDEFINED OR IN OVERLAY               | The stack address specified by the /M switch was either undefined or in an overlay. The stack address is set to the system default.  |
| ?SYMBOL TABLE OVERFLOW?                              | There were too many global symbols used in the program. Retry the link using the /S switch. If the error still occurs, the link cannot take place in the available memory. |
| ?TOO MANY OUTPUT FILES?                              | The Linker allows specification of only two output files.  |
| TRANSFER ADDRESS UNDEFINED OR IN OVERLAY             | The transfer address was not defined or was in an overlay.   |
| UNDEFINED GLOBALS<br>xxxxxx<br>xxxxxx<br>.<br>.<br>. | The globals listed (xxxxxx) were undefined. If a load map is requested, this condition also causes the warning message, UNDEF GLBLS, to be printed on the terminal.        |



## CHAPTER 7

### LIBRARIAN

The RT-11 system provides the user with the capability of maintaining libraries which may be composed of functions and routines of his choice. Each library is a file containing a library header, library directory (or entry point table), and one or more object modules. The object modules in a library file may be routines which are repeatedly used in a program, routines which are used by more than one program, or routines which are related and simply gathered together for ease in usage--the contents of the library file are determined by the user's needs. An example of a typical library file is the FORTRAN library, FORLIB.OBJ. This library is provided with the FORTRAN package and contains all the mathematical functions needed for normal usage.

Object modules in a library file are accessed from another program via calls to their entry points; the object modules are linked with the program which uses them by the Linker (Chapter 6) to produce a single load module.

The RT-11 Librarian (LIBR) allows the user to create, update, modify, list, and maintain library files.

#### 7.1 CALLING AND USING LIBR

The RT-11 Librarian is called from the system device by entering the command:

R LIBR

in response to the dot printed by the Keyboard Monitor. The Command String Interpreter prints an asterisk at the left margin on the console terminal when it is ready to accept a command line.

Type CTRL C to halt the Librarian at any time and return control to the monitor. To restart the Librarian, type R LIBR or the REENTER command in response to the monitor's dot.



## Librarian

### 7.2 USER SWITCH COMMANDS AND FUNCTIONS

The user maintains library files through the use of switch commands. Functions which can be performed include object module deletion, insertion and replacement, library file creation, and listing of a library file's contents.

#### 7.2.1 Command Syntax

LIBR accepts command strings in the following general format:

\*dev:lib,dev:list=dev:input/s1/s2/s3

where:

|         |  |
|---------|--|
| dev:    | represents a legal RT-11 device specification              |
| lib     | represents the library file to be created or updated       |
| list    | represents a listing file for the library's contents       |
| input   | represents the filenames of the input object modules       |
| /s1,... | represents one or more of the switches listed in Table 7-1 |

Devices and filenames are specified by the user in the standard RT-11 command string syntax, with default extensions assigned as follows:

| <u>File</u>   | <u>Extension</u> |
|---------------|------------------|
| list file:    | .LLD             |
| library file: | .OBJ             |
| input files:  | .OBJ             |

If no device is specified, the default device (DK:) is assumed.

Each input file is made up of one or more object modules, and is stored on a given device under a specific filename and extension. Once an object module has been inserted into a library file, the module is no longer referenced by the name of the file of which it was a part, but by its individual module name. (This module name has been assigned by the assembler either via a .TITLE statement in the assembly source program, or, if no .TITLE statement is present, with the default name .MAIN.; see Chapter 5.) Thus, for example, the input file FORT.OBJ may exist on DT2: and may contain an object module called ABC. Once the module is inserted into a library file, reference is made only to ABC (not FORT.OBJ).

#### 7.2.2 LIBR Switch Commands

Table 7-1 summarizes the switches available for use under RT-11 LIBR. Switches are explained in detail following the table.



Table 7-1  
LIBR Switches

| Switch | Position In Command String | Meaning  |
|--------|----------------------------|--|
| /C     | Any line but last          | Command continuation; the command is too long for the current line and is continued on the next line |
| /D     | 1st line only              | Delete; delete modules from a library file   |
| /G     | 1st line only              | Global deletion; delete entry points from the library directory                                      |
| /R     | 1st line only              | Replace; replace modules in a library file   |
| /U     | 1st line only              | Update; insert and replace modules in a library file   |

There is no switch to indicate module insertion. The function of inserting a module into a library file is assumed in the absence of other switches.

7.2.2.1 Command Continuation Switch - The Command Continuation switch is necessary whenever there is not enough room to enter a command string on one line and additional lines are needed. The /C switch is typed at the end of the current line and may be repeated at the end of subsequent command lines as often as necessary as long as memory is available; if memory is exceeded, an error message is output. A /C switch is not entered on the last line of input.

Command Format:

```
*dev:lib,dev:list=dev:input1,dev:input2,...,/C
*dev:inputn
```

where:

dev: represents a device specification

lib represents the filename of the library to be created or updated

list represents the filename of a listing file containing the library file's contents

input represents the filenames of the input modules to be inserted into the library

/C represents the Continuation switch, indicating that the command is to be continued on the following line



## Librarian

### Examples:

```
*ALIB,LIBLIST=DT1:MAIN,TEST,FXN/C
*DT1:TRACK
```

In this example, a library file is created on the default device (DK:) under the filename ALIB.OBJ; a listing of the library file's contents is created as LIBLIST.LLD also on the default device; the filenames of the input modules are MAIN.OBJ, TEST.OBJ, FXN.OBJ, and TRACK.OBJ, all from DT1.

```
*BLIB=MAIN/C
*RK1:TEST/C
*RK0:FXN/C
*DT1:TRACK
```

A library file is created on the default device, (DK:) under the name BLIB. No listing is produced. Input files are MAIN from the default device, TEST from RK1:, FXN from RK0: and TRACK from DT1:

Another way of writing this command line is:

```
*BLIB=MAIN,RK1:TEST,RK0:FXN/C
*DT1:TRACK
```

**7.2.2.2 Creating a Library File** - A library file is created whenever a filename is indicated on the output side of a command line which does not represent a list file.

### Command Format:

```
*dev:lib=dev:input1,...,dev:inputn
```

where:

|       |   |
|-------|---|
| dev:  | represents a device specification   |
| lib   | represents the filename of the library to be created                              |
| input | represents the filenames of the input modules to be inserted into the new library |

### Example:

```
*NEWLIB=FIRST,SECOND
```

A new library called NEWLIB.OBJ is created on the default device (DK:). The modules which will make up this library file are in the files FIRST.OBJ and SECOND.OBJ, both on the default device.

Assume this command line is next entered:

```
*NEWLIB,LIST=THIRD,FOURTH
```

The already existing library file NEWLIB is destroyed when the new library file is created. A listing of the library file's contents is created under the filename LIST, and the object modules in the files THIRD and FOURTH are inserted into the library file NEWLIB.



## Librarian

7.2.2.3 Inserting Modules Into a Library - The Insert function is assumed whenever an input file does not have an associated switch; the modules in the file are inserted into the library file named on the output side of the command string. Any number of input files are allowed. If an attempt is made to insert a file which contains an entry point or .CSECT having the same name as an entry point or .CSECT already existing in the library file, an error message is printed. The library file is not updated, and control returns to the CSI; the user may enter another command string. (Refer to Section 7.2.2.6 for a method of circumventing this condition.)

Although the user may insert object modules which exist under the same name (as assigned by the .TITLE statement) this practice is not recommended because of the difficulty involved when replacing or updating these modules (refer to Sections 7.2.2.4 and 7.2.2.7).

### NOTE

The library operations of module insertion, replacement, deletion, merge, and update are actually performed in conjunction with the library file creation operation. Therefore, the library file to which the operation is directed must be indicated on both the input and output sides of the command line, since effectively a "new" output library file is created each time the operation is performed. The library file must be specified first in the input field.

### Command Format:

\*dev:lib=dev:lib,dev:input1,...,dev:inputn

### where:

|       |  |
|-------|--|
| dev:  | represents a device specification  |
| lib   | represents the filename of an existing library file                          |
| input | represents the filenames of the modules to be inserted into the library file |

### Example:

\*DXY=DXY,DT1:FA,FB,FC

The modules included in the files FA.OBJ, FB.OBJ, and FC.OBJ on DT1: are inserted into a library file named DXY.OBJ on the default device. The library header and Entry Point Table of the library file are updated accordingly (see Section 7.4).

7.2.2.4 Replace Switch - The Replace function is used to replace modules in a library file. All modules contained in the file(s) indicated as input will replace existing modules of the same names in the library file specified as output.



## Librarian

An error message is printed and no modules are replaced if an old module does not exist under the same name as an input module, or if the user specifies the /R switch on a library file. /R must follow each input filename containing modules for replacement.

### Command Format:

```
*dev:lib=dev:lib,input1/R,...,dev:inputn/R
```

### where:

|       |   |
|-------|---|
| dev:  | represents a device specification                                   |
| lib   | represents the filename of an existing library file                 |
| input | represents the names of the files containing modules to be replaced |
| /R    | represents the Replace switch                                       |

### Examples:

```
*TFIL=TFIL,INA,INB/R,INC
```

This command line indicates that the modules in the file INB.OBJ are to replace existing modules of the same names in the library file TFIL.OBJ. The object modules in the files INA.OBJ and INC.OBJ are to be added. All files are stored on the default device DK:.

```
*XFIL=TFIL,INA,INB/R,INC
```

The same operation occurs here as in the preceding example, except that this updated library file is assigned the new name XFIL.

**7.2.2.5 Delete Switch** - The Delete switch deletes modules and all their associated entry points from the library.

### Command Format:

```
*dev:lib=dev:lib/D
```

### where:

|      |  |
|------|--|
| dev: | represents the device on which the library file exists   |
| lib  | represents the filename of an existing library file  |
| /D   | represents the Delete switch; may be positioned anywhere on the input side of the command line |

When the /D switch is used, the Librarian prints:

```
MOD NAME:
```



## Librarian

The user should respond with the name of the module to be deleted followed by a carriage return; he may continue until all modules to be deleted have been entered. Typing only a carriage return (either on a line by itself or immediately after the MOD NAME: message) terminates input and initiates execution of the command line.

### Examples:

```
*DT3:TRAP=DT3:TRAP/D
```

```
MOD NAME:  
SGN <CR>  
TAN <CR>  
<CR>
```

The modules SGN.OBJ and TAN.OBJ are deleted from the library file TRAP.OBJ on DT3:.

```
*LIBFIL=LIBFIL/D,ABC/R,DEF
```

```
MOD NAME:  
FIRST <CR>  
<CR>
```

The module FIRST.OBJ is deleted from the library (LIBFIL); the module ABC.OBJ replaces an old module of the same name in the library, and the modules in the file DEF.OBJ are inserted into the library.

```
*LIBFIL=LIBFIL/D
```

```
MOD NAME:  
X CR>  
X CR>  
<CR>
```

Two modules of the same name are deleted from the library file LIBFIL (module names are assigned with the .TITLE statement as described in Section 7.2.1).

**7.2.2.6 Delete Global Switch** - The Delete Global switch gives the user the ability to delete a specific entry point from a library file's Entry Point Table.

### Command Format:

```
*dev:lib=dev:lib/G
```

### where:

|      |  |
|------|--|
| dev: | represents the device on which the library file exists |
| lib  | represents the filename of an existing library file    |



Librarian

/G represents the Delete Global switch; may be positioned anywhere on the input side of the command line

When the /G switch is used, the Librarian prints:

ENTRY POINT:

The user should respond with the name of the entry point to be deleted followed by a carriage return; he may continue until all entry points to be deleted have been entered. Typing only a carriage return (either on a line by itself or immediately after the ENTRY POINT: message) terminates input and initiates execution of the command line.

Example:

```
*ROLL=ROLL/G
```

```
ENTRY POINT:
NAMEA <CR>
NAMEB <CR>
<CR>
```

This command instructs LIBR to delete the entry points NAMEA and NAMEB from the entry point table found in the library file ROLL.OBJ on DK:.

As mentioned in Section 7.2.2.3, if an attempt is made to insert modules into a library file when entry points within either the library file or any of the modules exist under duplicate names, an ?ILL INS? message is printed. To circumvent this condition, the /G switch can be used to delete all but one of each of the duplicate entry points.

For example, assume the modules A, B and C are to be inserted into a library file called XLIB. However, XLIB, A, B and C each contain an entry point called COM1, and A, B and C each contain an entry point called COM2. The following command line is entered:

```
*XLIB=XLIB,A,B,C/G
```

```
ENTRY POINT:
COM1 <CR>
COM1 <CR>
COM1 <CR>
COM2 <CR>
COM2 <CR>
<CR>
```

This deletes all but one each of the duplicate entry points, allowing the insert to take place.

Since entry points are only deleted from the Entry Point Table (and not from the library itself) whenever a library file is updated, all entry points that were previously deleted are restored unless the /G switch is again used to delete them. This feature allows the user to recover from inadvertently deleting the wrong entry point.



## Librarian

7.2.2.7 Update Switch - The Update switch allows the user to update a library file by combining the insert and replace functions. If the object modules included in an input file in the command line already exist in the library file, they are replaced; if not, they are inserted. (No error messages are printed when using the Update function as might occur under the Insert and Replace functions.) /U must follow each input file containing modules to be updated.

### Command Format:

```
*dev:lib=dev:lib,dev:input1/U,...,dev:inputn/U
```

### where:

|       |  |
|-------|--|
| dev:  | represents a device specification                                      |
| lib   | represents the filename of an existing library file                    |
| input | represents the names of files containing object modules to be updated. |
| /U    | represents the Update switch   |

### Examples:

```
*BALIB=BALIB,FOLT/U,TAL,BART/U
```

This command line instructs LIBR to update the library file BALIB.OBJ on the default device. First the modules in FOLT.OBJ and BART.OBJ replace old modules of the same names in the library file, or if none already exist under their names, the modules are inserted. Then the modules from the file TAL.OBJ are inserted; an error message is printed if the name of the module in TAL.OBJ already exists.

```
*XLIB=XLIB/D,Z/U/G
```

```
MOD NAME:  
X <CR>  
X <CR>  
<CR>
```

```
ENTRY POINT:  
SEC <CR>  
SEC1 <CR>  
<CR>
```

There are two object modules of the same name (X) in both Z and XLIB; these are first deleted from XLIB. This ensures that both modules X in file Z are correctly placed into the library. Entry points SEC and SEC1 are also deleted from the Entry Point Table, but automatically return when the library (XLIB) is updated.

7.2.2.8 Listing the Directory of a Library File - The user may specify that a listing of the contents of a library file be output by indicating both the library file and a list file in the command line. Since a library file is not being created or updated, it is not necessary to indicate the filename on the output side of the command line; however a comma must be used to designate a null output library file.



## Librarian

### Command Formats:

`*,LP:=dev:lib`  
or  
`*,dev:list=dev:lib`

### where:

`dev:` represents a device specification  
`lib` represents the file name of an existing library file  
`LP:` indicates the listing is to be sent directly to the line printer  
`list` represents a list file of the library file's contents

### Examples:

`*,DT2:LIST=LIBFIL`

This command line outputs to DECTape 2 as LIST.LLD a listing of the contents of the library file LIBFIL.OBJ on the default device.

`*,LP:=FLIB`

This command outputs on the line printer a listing of all modules in the library file FLIB.OBJ stored on the default device. Assuming this library is composed of modules STOP, WAIT, and IMUL, is 2 blocks long, was created on September 6, 1974, and the listing was requested on September 6, 1974, the directory format appears as follows:

|                 |             |             |             |
|-----------------|-------------|-------------|-------------|
| RT-11 LIBRARIAN | X02-05      | 6-SEP-74    |             |
| FLIB            | 6-SEP-74    | 2 BLOCKS    |             |
| MODULE          | ENTRY/CSECT | ENTRY/CSECT | ENTRY/CSECT |
| STOP            | STPS        |             |             |
| WAIT            | SWAIT       |             |             |
| IMUL            | MUISIS      | MUISMS      | MUISPS      |
|                 | MUISSS      | SMLI        |             |

7.2.2.9 Merging Library Files - Two or more library files may be merged under one filename by indicating all the library files to be merged in a single command line. The individual library files are not deleted following the merge.

### Command Format:

`*dev:lib=dev:input1,...,dev:inputn`

### where:

`dev:` represents a device specification  
`lib` represents the name of the library file which will contain all the merged files (if a library file



## Librarian

already exists under this name, it must also be indicated in the input side of the command line in order to be included in the merge)

input represents the library files to be merged together

Thus, the command:

```
*MAIN=MAIN,TRIG,STP,BAC
```

combines library files MAIN.OBJ, TRIG.OBJ, STP.OBJ, and BAC.OBJ under the existing library file name MAIN.OBJ; all files are on the default device DK:.

```
*FORT=A,B,C
```

This command creates a library file named FORT.OBJ and merges existing library files A.OBJ, B.OBJ, and C.OBJ under the filename FORT.OBJ.

### NOTE

Library files that have been combined under PIP are illegal as input to both the Librarian and the Linker.

## 7.3 COMBINING LIBRARY SWITCH FUNCTIONS

Two or more library functions may be requested in the same command line. The Librarian performs functions in the following order:

1. /C
2. /D
3. /G
4. /U
5. /R
6. Insertions
7. Listing

Example:

```
*FILE,LP:=FILE/D,MODX,MODY/R
```

```
MOD NAME:  
XYZ<CR>  
A<CR>  
<CR>
```

Functions in this example are performed in order, as follows:

1. Delete modules XYZ.OBJ and A.OBJ from the library file FILE.OBJ
2. Replace any duplicate of the module in the file MODY.OBJ
3. Insert the modules in the file MODX.OBJ
4. List the contents of FILE.OBJ on the line printer



## Librarian

### 7.4 FORMAT OF LIBRARY FILES

A library file is a contiguous file consisting of a header, an Entry Point Table (library directory) and one or more library object modules, as illustrated in Figure 7-1:

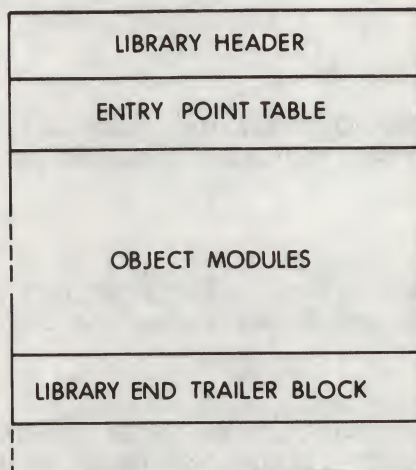


Figure 7-1  
General Library File Format

The following paragraphs describe in detail each component of a library file.

#### 7.4.1 Library Header

The header section of a library file contains 17 (decimal) words which describe the current status of the file (refer to Figure 7-2). This includes information relating to the version of the Librarian in use, the date and time of file creation or update, the relative starting address of the Entry Point Table (EPT), the number of EPT entries available and in use, and the placing of the next module to be inserted into the library file. The contents of the library header are updated as the library file is modified, so that LIBR can always quickly and easily access the information it needs to perform its functions. Figure 7-2 illustrates the header format.



## Librarian

|                 |   |
|-----------------|---|
| 1               | } FORMATTED BINARY<br>BLOCK HEADER            |
| 56 <sub>8</sub> |   |
| 7               | LIBRARIAN CODE                                |
| 1               | VERSION NUMBER                                |
| 0               |   |
| 0               | YEAR-MONTH-DAY                                |
| 0               |   |
| 0               |   |
| 0               |   |
| 0               |   |
| 0               |   |
| 12 <sub>8</sub> | EPT RELATIVE START ADDRESS                    |
| X1              | EPT ENTRIES ALLOCATED IN BYTES                |
| 0               | EPT ENTRIES AVAILABLE (NOT USED IN VERSION 1) |
| X2              | NEXT INSERT RELATIVE BLOCK NUMBER             |
| X3              | NEXT BYTE WITHIN BLOCK                        |
| 0               | NOT USED                                      |

Figure 7-2  
Library Header Format

### 7.4.2 Entry Point Table (Library Directory)

The Entry Point Table is located immediately after the library header. It is composed of four-word entries which include the names, addresses, and entry points of all object modules in the library file. The first two words of an entry in the EPT contain the Radix 50 name by which a module is referenced. The third word provides a pointer to the object module where an entry point is defined. The fourth word contains the total number of CSECTs in the object module (information needed by the Linker), and the relative byte within the block pointing to the object module's starting point, as shown in Figure 7-3.

The EPT is accessed sequentially from top to bottom whenever a library module is called by either its module name or an entry point.

|   |                                 |                        |  |
|---|---------------------------------|------------------------|--|
| 0 | SYMBOL (RAD 50)                 |                        |  |
| 2 | SYMBOL (RAD 50)                 |                        |  |
| 4 |                                 | ADDRESS OF BLOCK       | BIT 15=1-MODULE NAME   |
| 6 | # OF CSECTS IN<br>OBJECT MODULE | RELATIVE BYTE IN BLOCK | RELATIVE BYTE MAXIMUM=777 <sub>8</sub><br>CSECTS MAXIMUM =177 <sub>8</sub> |

Figure 7-3  
Format of Entry Point Table



## Librarian

### 7.4.3 Object Modules

Object modules follow the Entry Point Table. An object module consists of four main types of data blocks: a global symbol directory, text blocks, a relocation directory, and an internal symbol directory. The information contained in these data blocks is used by the Linker during creation of a load module.

### 7.4.4 Library End Trailer

Following all object modules in a library file is a specially coded library end trailer which signifies the end of the file. This trailer is illustrated in Figure 7-4.

|    |                         |
|----|-------------------------|
| 1  | FORMATTED BINARY HEADER |
| 10 | FORMATTED BINARY LENGTH |
| 10 | TYPE CODE               |
| 0  | NOT USED                |

Figure 7-4  
Library End Trailer

### 7.5 LIBR ERROR MESSAGES

The following error messages are printed following incorrect use of LIBR; if any errors result during library processing, the user must reenter the command.

| <u>Message</u> | <u>Meaning</u>   |
|----------------|--|
| ?BAD LIBR?     | The user has attempted to build a library file containing no directory entries or he has given an illegally constructed library file to the Librarian as input.                |
| ?BAD OBJ?      | A bad object module was detected during input.   |
| ?CSECT ERROR?  | The user has extended beyond the allowable .CSECT space for an object module to be placed in the library (i.e., the object module contains greater than 127(decimal) .CSECTs). |



Librarian

Message

Meaning

?DEV FULL?

The device is full; LIBR is unable to create or update the indicated library file. The CSI prints an asterisk and waits for the user to enter another command line.

?FIL NOT FND?

One of the input files indicated in the command line was not found. The CSI prints an asterisk; the command may be reentered.

?ILL CMD?

An illegal command was used in the command line. The CSI prints an asterisk; the command may be reentered.

xxxxxx ?ILL DEL?

An attempt was made to delete from the library's directory a module or an entry point that does not exist; xxxxxx represents the module or entry point name. The CSI prints an asterisk and waits for the user to enter another command line.

?ILL DEV?

An illegal device was specified in the command line. The CSI prints an asterisk; the command may be reentered.

xxxxxx ?ILL INS?

An attempt was made to insert a module into a library which contains the same entry point as an existing module. xxxxxx represents the entry point name. The CSI prints an asterisk; a new command may be entered.

xxxxxx ?ILL REPL?

An attempt was made to replace in the library file a module which does not already exist. xxxxxx represents the module name. The CSI prints an asterisk and waits for the user to enter another command line.

?IN ERR?

An unrecoverable hardware/software error has occurred while processing an input file. The CSI prints an asterisk and waits for another command to be entered.

?LIBR FIL ILL REPL?

The user has specified that a library file be replaced by another library file. Only object modules can be replaced.

?NO CORE?

Available free memory has been used up. The current command is aborted and the CSI prints an asterisk; a new command may be entered.

?OUT ERR?

An unrecoverable hardware/software error has occurred while processing an output file. The CSI prints an asterisk and waits for the user to enter another command.







## CHAPTER 8

### ON-LINE DEBUGGING TECHNIQUE

RT-11 On-line Debugging Technique (ODT) is a system program that aids in debugging assembled and linked object programs. From the keyboard, the user interacts with ODT and the object program to:

1. Print the contents of any location for examination or alteration.
2. Run all or any portion of an object program using the breakpoint feature.
3. Search the object program for specific bit patterns.
4. Search the object program for words which reference a specific word.
5. Calculate offsets for relative addresses.
6. Fill a single word, block of words, byte or block of bytes with a designated value.

The assembly listing of the program to be debugged should be readily available when ODT is being used. Minor corrections to the program can be made on-line during the debugging session, and the program may then be run under control of ODT to verify any changes made. Major corrections, however (such as a missing subroutine), should be noted on the assembly listing and incorporated in a subsequent updated program assembly.

#### 8.1 CALLING AND USING ODT

ODT is supplied as a relocatable object module. It can be linked with the user program (using the RT-11 Linker) for an absolute area in memory and loaded with the user program.

Once loaded in memory with the user program, ODT has three legal start or restart addresses. The lowest (O.ODT) is used for normal entry, retaining the current breakpoints. The next (O.ODT+2) is a restart address which clears all breakpoints and re-initializes ODT saving the general registers and clearing the relocation registers. The last address (O.ODT+4) is used to reenter ODT. A reenter saves the Processor Status and general registers and removes the breakpoint



## On-Line Debugging Technique

instructions from the user program. ODT prints the Bad Entry (BE) error message. Breakpoints which were set are reset on the next ;G command. (;P is illegal after a BE message.) The ;G and ;P commands are used to run a program and are explained in Section 8.3.7.

The absolute address used is the address of the entry point 0.ODT shown in the Linker load map. 0.ODT is always the lowest address of ODT+172, i.e., 0.ODT is relative location 172 in ODT.

### NOTE

If linked with an overlay structured file, ODT should reside in the root segment so it is always in memory. A breakpoint inserted in an overlay will be destroyed if it is overlaid during program execution.

If ODT is being used in a Foreground/Background environment with another job running, ODT's priority bit must be set to 0 as follows:

```
*$P/000007 0 <CR>
```

This puts ODT into the wait state at level 0, not 7. If this is not done, all interrupts (including clock) will be locked out while ODT is waiting for terminal input.

### Examples:

#### 1. ODT Linked with the User Program:

```
_ GET USER.SAV
_ START 1172
_ ODT V01-01
_
```

User program previously linked to ODT is brought into memory. Value (1172) of entry point 0.ODT (determined from Linker load map) is used to start ODT.

#### 2. Loading ODT with the User Program:

```
_ GET USER.SAV
_ GET ODT.SAV
_ START 1172
_ ODT V01-01
_
```

User program is loaded into memory.  
ODT is loaded into memory.  
Assuming ODT has been linked for a bottom address of 1000, ODT starts.

#### 3. Restarting ODT Clearing Breakpoints:

```
_ START 1174
_
```

Assuming ODT was originally linked for a bottom address of 1000, this command (0.ODT+2) re-initializes ODT and clears any previous breakpoints.



## On-Line Debugging Technique

#### 4. Reentering ODT:

. START 1176

BE001212

✱

Assuming ODT was linked for a bottom address of 1000, the value of 0.ODT 1172+4 is used as the start address.

## 5. Using ODT with Foreground/Background Jobs:

It is possible to use ODT to debug programs written as either background or foreground jobs. In the background or under the Single-Job Monitor, ODT can be linked with the program as described in Example 1 above.

To debug a program in the foreground area, it is recommended that ODT be run in the background while the program to be debugged is in the foreground. The sequence of commands to do this is:

```

.FRUN PROG/P
LOADED AT xxxxxx
.RUN ODT

```

```

ODT  V01-01
*xxxxxx;OR
*$F/000000 0<CR>
*0;G

```

. RSU

Load the foreground program.  
The first address of the job is  
printed (xxxxxxx)  
Run ODT in the background  
and set a relocation register  
to the start of the job. \$F  
is the format register. It  
should be cleared to enable proper  
address print out. 0;G starts the  
Keyboard Monitor again, and .RSU  
starts the foreground job.

The copy of ODT used must be linked low enough so that it will fit in memory along with the foreground job.

## NOTE

Since ODT uses its own terminal handler, it cannot be used with the display hardware. If GT ON has been typed, ODT will ignore it and direct I/O only to the console terminal.

### 8.1.1 Return to Monitor, CTRL C

If ODT is awaiting a command, a CTRL C from the keyboard calls the RT-11 Keyboard Monitor. The monitor responds with a ↑C on the terminal and awaits a Keyboard Monitor command. (The monitor REENTER command may be used to reenter ODT only if the user program has set the reenter bit. Otherwise ODT is reentered at address 0.ODT+4 as shown above.)



## On-Line Debugging Technique

### 8.1.2 Terminate Search, CTRL U

If typed during a search printout, a CTRL U terminates the search and ODT prints an asterisk.

## 8.2 RELOCATION

When the assembler produces a binary object module, the base address of the module is taken to be location 000000, and the addresses of all program locations as shown in the assembly listing are indicated relative to this base address. After the module is linked by the Linker, many values within the program, and all the addresses of locations in the program, will be incremented by a constant whose value is the actual absolute base address of the module after it has been relocated. This constant is called the relocation bias for the module. Since a linked program may contain several relocated modules each with its own relocation bias, and since, in the process of debugging, these biases will have to be subtracted from absolute addresses continually in order to relate relocated code to assembly listings, RT-11 ODT provides an automatic relocation facility.

The basis of the relocation facility lies in eight relocation registers, numbered 0 through 7, which may be set to the values of the relocation biases at different times during debugging. Relocation biases should be obtained by consulting the memory map produced by the Linker. Once set, a relocation register is used by ODT to relate relocatable code to relocated code. For more information on the exact nature of the relocation process, consult Chapter 6, the RT-11 Linker.

### 8.2.1 Relocatable Expressions

A relocatable expression is evaluated by ODT as a 16-bit (6-digit octal) number and may be typed in any one of the three forms presented in Table 8-1. In this table, the symbol *n* stands for an integer in the range 0 to 7 inclusive, and the symbol *k* stands for an octal number up to six digits long, with a maximum value of 177777. If more than six digits are typed, ODT takes the last six digits, truncated to the low-order 16 bits. *k* may be preceded by a minus sign, in which case its value is the two's complement of the number typed. For example:

| <u>k (number typed)</u> | <u>Values</u> |
|-------------------------|---------------|
| 1                       | 000001        |
| -1                      | 177777        |
| 400                     | 000400        |
| -177730                 | 000050        |
| 1234567                 | 034567        |



## On-Line Debugging Technique

Table 8-1  
Forms of Relocatable Expressions (r)

|    | r                               | Value of r   |
|----|---------------------------------|--|
| A) | k                               | The value of r is simply the value of k.   |
| B) | n,k                             | The value of r is the value of k plus the contents of relocation register n. If the n part of this expression is greater than 7, ODT uses only the last octal digit of n.  |
| C) | C or<br>C,k or<br>n,C or<br>C,C | Whenever the letter C is typed, ODT replaces C with the contents of a special register called the Constant Register. This value has the same role as the k or n that it replaces (i.e., when used in place of n it designates a relocation register). The Constant Register is designated by the symbol \$C and may be set to any value, as indicated below. |

In the following examples, assume in each case that relocation register 3 contains 003400 and that the constant register contains 000003.

| <u>r</u> | <u>Value of r</u> |
|----------|-------------------|
| 5;C      | 000005            |
| -17;C    | 177761            |
| 3,0;C    | 003400            |
| 3,150;C  | 003550            |
| 3,-1;C   | 003377            |
| C;C      | 000003            |
| 3,C;C    | 003403            |
| C,0;C    | 003400            |
| C,10;C   | 003410            |
| C,C;C    | 003403            |

### NOTE

For simplicity most examples in this section use Form A. All three forms of r are equally acceptable, however.

## 8.3 COMMANDS AND FUNCTIONS

When ODT is started (as explained in Section 8.1) it indicates readiness to accept commands by printing an asterisk on the left margin of the terminal page. Most of the ODT commands can be issued in response to the asterisk. For example, a word can be examined and changed if desired, the object program can be run in its entirety or in segments, or memory can be searched for certain words or references to certain words. The discussion below explains these features. In the following examples, characters output by ODT are underlined to differentiate from user input.

### 8.3.1 Printout Formats

Normally, when ODT prints addresses (as with the commands ↓, ↑, ←, @, <, and >) it attempts to print them in relative form (Form B in Table



## On-Line Debugging Technique

8-1). ODT looks for the relocation register whose value is closest but less than or equal to the address to be printed, and then represents the address relative to the contents of the relocation register. However, if no relocation register fits the requirement, the address is printed in absolute form. Since the relocation registers are initialized to -1 (the highest number) the addresses are initially printed in absolute form. If any relocation register subsequently has its contents changed, it may then, depending on the command, qualify for relative form.

For example, suppose relocation registers 1 and 2 contain 1000 and 1004 respectively, and all other relocation registers contain numbers much higher. Then the following sequence might occur (the slash command causes the contents of the location to be printed; the line feed command (<LF>) accesses the next sequential location):

```
*774/000000 <LF>
000776 /000000 <LF>
1,000000 /000000 <LF>      (absolute location 1000)
1,000002 /000000 <LF>      (absolute location 1002)
2,000000 /000000          (absolute location 1004)
```

The printout format is controlled by the format register, \$F. Normally this register contains 0, in which case ODT prints addresses relatively whenever possible. \$F may be opened and changed to a non-zero value, however, in which case all addresses will be printed in absolute form (see paragraph 8.3.4, Accessing Internal Registers).

### 8.3.2 Opening, Changing, and Closing Locations

An open location is one whose contents ODT prints for examination, making those contents available for change. In a closed location, the contents are no longer available for change. Several commands are used for opening and closing locations.

Any command used to open a location when another location is already open causes the currently open location to be closed. The contents of an open location may be changed by typing the new contents followed by a single-character command which requires no argument (i.e., <LF>, ↑, RETURN, ←, @, >, <).

#### The Slash, /

One way to open a location is to type its address followed by a slash:

```
*1000/012746
```

Location 1000 is open for examination and is available for change.

If the contents of an open location are not to be changed, type the RETURN key and the location is closed; ODT prints an asterisk and waits for another command. However, to change the word, simply type the new contents before giving a command to close the location:

```
*1000/012746 012345 <CR>
*
```



## On-Line Debugging Technique

In the example above, location 1000 now contains 012345 and is closed since the RETURN key was typed after entering the new contents, as indicated by ODT's second asterisk.

Used alone, the slash reopens the last location opened:

```
*1000/012345 2340 <CR>  
*/002340
```

In the example above, the open location was closed by typing the RETURN key. ODT changed the contents of location 1000 to 002340 and then closed the location before printing the \*. The single slash command directed ODT to reopen the last location opened. This allowed verification that the word 002340 was correctly stored in location 1000.

Note again, that opening a location while another is open automatically closes the currently open location before opening the new location.

Also note that if an odd numbered address is specified with a slash, ODT opens the location as a byte, and subsequently behaves as if a backslash had been typed (see the following paragraph).

### The Backslash, \

In addition to operating on words, ODT operates on bytes. One way to open a byte is to type the address of the byte followed by a backslash. (On the LT33 or LT35 terminal \ is typed by pressing the SHIFT key while typing the L key.) This causes not only the printing of the byte value at the specified address but also the interpreting of the value as ASCII code, and the printing of the corresponding character (if possible) on the terminal:

```
*1001\101 =A
```

A backslash typed alone reopens the last open byte. If a word was previously open, the backslash reopens its even byte:

```
*1002/000004 \004 =
```

### The LINE FEED Key, <LF>

If the LINE FEED key is typed when a location is open, ODT closes the open location and opens the next sequential location:

```
*1000/002340 <LF> ( <LF> denotes typing the LINE FEED key)  
001002 /012740
```

In this example, the LINE FEED key caused ODT to print the address of the next location along with its contents, and to wait for further instructions. After the above operation, location 1000 is closed and 1002 is open. The open location may be modified by typing the new contents.

If a byte location was open, typing the LINE FEED key opens the next byte location.



## On-Line Debugging Technique

### The Up-Arrow, ↑ or ^

If the up-arrow (or circumflex) is typed when a location is open (up-arrow is produced on an LT33 or LT35 by typing SHIFT N), ODT closes the open location and opens the previous location. To continue from the example above:

```
*001002/012740 ↑  
001000 /002340
```

Now location 1002 is closed and 1000 is open. The open location may be modified by typing the new contents.

If the opened location was a byte, then up-arrow opens the previous byte.

### The Back-Arrow, ← or \_

If the back-arrow (or underline) is typed (via SHIFT O on an LT33 or LT35 terminal) to an open word, ODT interprets the contents of the currently open word as an address indexed by the Program Counter (PC) and opens the addressed location:

```
*1006/000006 ←  
001016 /000405
```

Notice in this example that the open location, 1006, was indexed by the PC as if it were the operand of an instruction with address mode 67 as explained in Chapter 5.

A modification to the opened location can be made before a line feed, up-arrow, or back-arrow is typed. Also, the new contents of the location will be used for address calculations using the back-arrow command. Example:

```
*100/000222 4 <LF> (modify to 4 and open next location)  
000102 /000111 6↑ (modify to 6 and open previous location)  
000100 /000004 100← (change to 100 and open location indexed  
000202 /123456 by PC)
```

### Open the Addressed Location, @

The at symbol @ (SHIFT P on the LT33 or LT35 terminal) may be used to optionally modify a location, close it, and then use its contents as the address of the location to open next.

```
*1006/001044 @ (open location 1044 next)  
001044 /000500
```

```
*1006/001044 2100@ (modify to 2100 and open location  
002100 /000167 2100)
```



## On-Line Debugging Technique

### Relative Branch Offset, >

The right-angle bracket, >, will optionally modify a location, close it, and then use its low-order byte as a relative branch offset to the next word to be opened. For example:

```
*1032/000407 301> (modify to 301 and interpret as a  
000636 /000010 relative branch)
```

Note that 301 is a negative offset (-77). The offset is doubled before it is added to the PC; therefore,  $1034 + (-176) = 636$ .

### Return to Previous Sequence, <

The left-angle bracket, <, allows the user to optionally modify a location, close it, and then open the next location of the previous sequence which was interrupted by a back-arrow, @, or right-angle bracket command. Note that back-arrow, @, or right-angle bracket causes a sequence change to the word opened. If a sequence change has not occurred, the left-angle bracket simply opens the next location as a LINE FEED does. This command operates on both words and bytes.

```
*1032/000407 301> (> causes a sequence change)  
000636 /000010 < (return to original sequence)  
001034 /001040 @ (@ causes a sequence change)  
001040 /000405 \005 = < (< now operates on byte)  
001035 \002 = < (< acts like <LF>)  
001036 \004 =
```

### 8.3.3 Accessing General Registers 0-7

The program's general registers 0-7 are opened with a command in the following format:

```
*$n/
```

where n is the integer representing the desired register (in the range 0 through 7). When opened, these registers can be examined or changed by typing in new data as with any addressable location. For example:

```
*$0/000033 <CR> (R0 was examined and closed)  
*
```

```
*$4/000474 464 <CR> (R4 was opened, changed, and closed)  
*
```

The example above can be verified by typing a slash in response to ODT's asterisk:

```
* /000464
```

The LINE FEED, up-arrow, back-arrow or @ command may be used when a register is open.



## On-Line Debugging Technique

### 8.3.4 Accessing Internal Registers

The program's Status Register contains the condition codes of the most recent operational results and the interrupt priority level of the object program. It is opened by typing `$S`. For example:

```
*$5/000311
```

`$S` represents the address of the Status Register. In response to `$S` in the example above, ODT printed the 16-bit word, of which only the low-order eight bits are meaningful. Bits 0-3 indicate whether a carry, overflow, zero, or negative (in that order) has resulted, and bits 5-7 indicate the interrupt priority level (in the range 0-7) of the object program. (Refer to the PDP-11 PROCESSOR HANDBOOK for the Status Register format.)

The `$` is used to open certain other internal locations listed in Table 8-2:

Table 8-2  
Internal Registers

| Register         | Function  |
|------------------|---|
| <code>\$B</code> | location of the first word of the breakpoint table (see Section 8.3.6).                                       |
| <code>\$M</code> | mask location for specifying which bits are to be examined during a bit pattern search (see Section 8.3.9).   |
| <code>\$P</code> | location defining the operating priority of ODT (see Section 8.3.15).   |
| <code>\$S</code> | location containing the condition codes (bits 0-3) and interrupt priority level (bits 5-7) (explained above). |
| <code>\$C</code> | location of the Constant Register (see Section 8.3.10).   |
| <code>\$R</code> | location of Relocation Register 0, the base of the Relocation Register table (see Section 8.3.13).            |
| <code>\$F</code> | location of Format Register (see Section 8.3.1).  |

### 8.3.5 Radix 50 Mode, X

The Radix 50 mode of packing certain ASCII characters three to a word is employed by many DEC-supplied PDP-11 system programs, and may be employed by any programmer via the MACRO Assembler's `".RAD50"` directive. ODT provides a method for examining and changing memory words packed in this way with the `X` command.

When a word is opened and the `X` command is typed, ODT converts the contents of the opened word to its 3-character Radix 50 equivalent and prints these characters on the terminal. One of the responses in Table 8-3 can then be typed:



## On-Line Debugging Technique

Table 8-3  
Radix 50 Terminators

| Response   | Effect   |
|--|--|
| RETURN key <CR>  | Closes the currently open location.  |
| LINE FEED key <LF>   | Closes the currently open location and opens the next one in sequence.     |
| ↑ key  | Closes the currently open location and opens the previous one in sequence. |
| Any three characters whose octal code is 040 (space) or greater. | Converts the three specified characters into packed Radix 50 format.       |
| Legal Radix 50 characters for this last response are:            |  |
| .      \$      Space      0 through 9      A through Z           |  |

If any other characters are typed, the resulting binary number is unspecified (that is, no error message is printed and the result is unpredictable). Exactly three characters must be typed before ODT resumes its normal mode of operation. After the third character is typed, the resulting binary number is available to be stored in the opened location by closing the location in any one of the ways listed in Table 8-3. Example:

```
*1000/042431 X=KBI CBA <CR>  
*1000/011421 X=CBA
```

### NOTE

After ODT has converted the three characters to binary, the binary number can be interpreted in one of many different ways, depending on the command which follows. For example:

```
*1234/063337 X=PRO_XIT/013704
```

Since the Radix 50 equivalent of XIT is 113574, the final slash in the example will cause ODT to open location 113574 if it is a legal address. (Refer to paragraph 8.5 for a discussion of command legality and detection of errors.)

### 8.3.6 Breakpoints

The breakpoint feature facilitates monitoring the progress of program execution. A breakpoint may be set at any instruction which is not referenced by the program for data. When a breakpoint is set, ODT replaces the contents of the breakpoint location with a trap instruction so that program execution is suspended when a breakpoint



## On-Line Debugging Technique

is encountered. The original contents of the breakpoint location are restored, and ODT regains control.

With ODT, up to eight breakpoints, numbered 0 through 7, can be set at any one time. A breakpoint is set by typing the address of the desired location of the breakpoint followed by ;B. Thus r;B sets the next available breakpoint at location r. (If all 8 breakpoints have been set, ODT ignores the r;B command.) Specific breakpoints may be set or changed by the r;nB command where n is the number of the breakpoint. For example:

```
*1020;B      (sets breakpoint 0)
_1030;B      (sets breakpoint 1)
*1040;B      (sets breakpoint 2)
_1032;1B     (resets breakpoint 1)
*
_
```

The ;B command removes all breakpoints. Use the ;nB command to remove only one of the breakpoints, where n is the number of the breakpoint. For example:

```
*;2B        (removes the second breakpoint)
*
```

A table of breakpoints is kept by ODT and may be accessed by the user. The \$B/ command opens the location containing the address of breakpoint 0. The next seven locations contain the addresses of the other breakpoints in order, and can be sequentially opened using the LINE FEED key. For example:

```
*$B/001020 <LF>
nnnnnn/001032 <LF>
nnnnnn/nnnnnn (nnnnnn=address internal to ODT)
```

In this example, breakpoint 2 is not set. The contents printed is an address internal to ODT and can be determined by checking the Linker Load Map (see Chapter 6).

It should be noted that a repeat count in a Proceed command refers only to the breakpoint that has most recently occurred. Execution of other breakpoints encountered is determined by their own repeat counts.

### 8.3.7 Running the Program, r;G and r;P

Program execution is under control of ODT. There are two commands for running the program: r;G and r;P. The r;G command is used to start execution (Go) and r;P to continue (Proceed) execution after halting at a breakpoint. For example:

```
*1000;G
```

Execution is started at location 1000. The program runs until a breakpoint is encountered or until program completion, unless it gets caught in an infinite loop, in which case it must be either restarted or reentered as explained in Section 8.1.

Upon execution of either the r;G or r;P command, the general registers 0-6 are set to the values in the locations specified as \$0-\$6 and the



## On-Line Debugging Technique

processor Status Register is set to the value in the location specified as \$S.

When a breakpoint is encountered, execution stops and ODT prints Bn; (where n is the breakpoint number), followed by the address of the breakpoint. Locations can then be examined for expected data. For example:

```
*1010;3B      (breakpoint 3 is set at location 1010)
*1000;G      (execution started at location 1000)
B3;001010   (execution stopped at location 1010)
*
```

To continue program execution from the breakpoint, type ;P in response to ODT's last \*.

When a breakpoint is set in a loop, it may be desirable to allow the program to execute a certain number of times through the loop before recognizing the breakpoint. This can be done by setting a proceed count using the k;P command; this command specifies the number of times the breakpoint is to be encountered before program execution is suspended (on the kth encounter). The count, k, refers only to the numbered breakpoint which most recently occurred. A different proceed count may be specified for the breakpoint when it is encountered. Thus:

```
B3;001010   (execution halted at breakpoint 3)
*1026;3B     (reset breakpoint 3 at location 1026)
*4;P         (set proceed count to 4 and
B3;001026   continue execution; loop through
*            breakpoint three times and halt on
             fourth occurrence of the breakpoint)
```

Following the table of breakpoints (as explained in Section 8.3.6) is a table of proceed command repeat counts for each breakpoint. These repeat counts can be inspected by typing \$B/ and nine LINE FEEDs. The repeat count for breakpoint 0 is printed (the first seven LINE FEEDs cause the table of breakpoints to be printed; the eighth types the single instruction mode, explained in the next section, and the ninth LINE FEED begins the table of proceed command repeat counts). The repeat counts for breakpoints 1 through 7, and the repeat count for the single-instruction trap follow in sequence. Before a proceed count is assigned a value by the user, it is set to 0; after the count has been executed, it is set to -1. Opening any one of these provides an alternative way of changing the count as the location, once open, can have its contents modified in the usual manner by typing the new contents and then the RETURN key. For example:

```
      :
nnnnnn /001036 <LF>      (address of breakpoint 7)
nnnnnn /006630 <LF>      (single instruction address)
nnnnnn /000000 15 <LF>   (count for breakpoint 0; change to 15)
nnnnnn /000000 <LF>     (count for breakpoint 1)
      :
      :
nnnnnn /000000 <LF>     (count for breakpoint 7)
nnnnnn /nnnnnn          (repeat count for single instruction
                        mode; the single instruction address
```



## On-Line Debugging Technique

is an address internal to the user program if single instruction mode is used)

The address indicated as the single instruction address and the repeat count for single instruction mode are explained next.

### 8.3.8 Single Instruction Mode

With this mode the number of instructions to be executed before suspension of the program run can be specified. The Proceed command, instead of specifying a repeat count for a breakpoint encounter, specifies the number of succeeding instructions to be executed. Note that breakpoints are disabled when single instruction mode is operative.

Commands for single instruction mode are:

|                  |   |
|------------------|---|
| <code>;nS</code> | Enables single instruction mode (n can have any non-zero value and serves only to distinguish this form from the form <code>;S</code> ). Breakpoints are disabled.  |
| <code>n;P</code> | Proceeds with program run for next n instructions before reentering ODT (if n is missing, it is assumed to be 1). Trap instructions and associated handlers can affect the Proceed repeat count. See Section 8.4.2. |
| <code>;S</code>  | Disables single instruction mode.   |

When the repeat count for single instruction mode is exhausted and the program suspends execution, ODT prints:

`B8;nnnnnn`

where nnnnnn is the address of the next instruction to be executed. The \$B breakpoint table contains this address following that of breakpoint 7. However, unlike the table entries for breakpoints 0-7, direct modification has no effect.

Similarly, following the repeat count for breakpoint 7 is the repeat count for single instruction mode. This table entry may be directly modified and thus is an alternative way of setting the single-instruction mode repeat count. In such a case, `;P` implies the argument set in the \$B repeat count table rather than an assumed 1.

### 8.3.9 Searches

With ODT all or any specified portion of memory can be searched for any specific bit pattern or for references to a particular location.



## On-Line Debugging Technique

### Word Search, r;W

Before initiating a word search, the mask and search limits must be specified. The location represented by \$M is used to specify the mask of the search. \$M/ opens the mask register. The next two sequential locations (opened by LINE FEEDs) contain the lower and upper limits of the search. Bits set to 1 in the mask are examined during the search; other bits are ignored. Then the search object and the initiating command are given using the r;W command where r is the search object. When a match is found, (i.e., each bit set to 1 in the search object is set to 1 in the word being searched over the mask range) the matching word is printed. For example:

|                                 |                              |
|---------------------------------|------------------------------|
| <u>*\$M/000000</u> 177400 <LF>  | (test high-order eight bits) |
| <u>nnnnnn /000000</u> 1000 <LF> | (set low address limit)      |
| <u>nnnnnn /000000</u> 1040 <CR> | (set high address limit)     |
| <u>*400;W</u>                   | (initiate word search)       |
| <u>001010 /000770</u>           |                              |
| <u>001034 /000404</u>           |                              |
| <u>*</u>                        |                              |

In the above example, nnnnnn is an address internal to ODT; this location varies and is meaningful only for reference purposes. In the first line above, the slash was used to open \$M which now contains 177400; the LINE FEEDs opened the next two sequential locations which now contain the upper and lower limits of the search.

In the search process an exclusive OR (XOR) is performed with the word currently being examined and the search object, and the result is ANDed to the mask. If this result is zero, a match has been found and is reported on the terminal. Note that if the mask is zero, all locations within the limits are printed.

Typing CTRL U during a search printout terminates the search.

### Effective Address Search, r;E

ODT provides a search for words which address a specified location. Open the mask register only to gain access to the low and high limit registers. After specifying the search limits (as explained for the word search), type the command r;E (where r is the effective address) to initiate the search.

Words which are either an absolute address (argument r itself), a relative address offset, or a relative branch to the effective address, are printed after their addresses. For example:

|                                 |                                  |
|---------------------------------|----------------------------------|
| <u>*\$M/177400</u> <LF>         | (open mask register only to gain |
| <u>nnnnnn /001000</u> 1010 <LF> | access to search limits)         |
| <u>nnnnnn /001040</u> 1060 <CR> |                                  |
| <u>*1034;E</u>                  | (initiating search)              |
| <u>001016 /001006</u>           | (relative branch)                |
| <u>001054 /002767</u>           | (relative branch)                |
| <u>*1020;E</u>                  | (initiating a new search)        |
| <u>001022 /177774</u>           | (relative address offset)        |
| <u>001030 /001020</u>           | (absolute address)               |



## On-Line Debugging Technique

Particular attention should be given to the reported effective address references because a word may have the specified bit pattern of an effective address without actually being so used. ODT reports all possible references whether they are actually used as such or not.

Typing CTRL U during a search printout terminates the search.

### 8.3.10 The Constant Register, r;C

It is often desirable to convert a relocatable address into its value after relocation or to convert a number into its two's complement, and then to store the converted value into one or more places in a program. The Constant Register provides a means of accomplishing this and other useful functions.

When r;C is typed, the relocatable expression r is evaluated to its 6-digit octal value and is both printed on the terminal and stored in the Constant Register. The contents of the Constant Register may be invoked in subsequent relocatable expressions by typing the letter C. Examples follow:

|                             |   |
|-----------------------------|---|
| *-4432;C= <u>173346</u>     | (the two's complement of 4432 is placed in the Constant Register)                                 |
| *6632/ <u>062701</u> C <CR> | (the contents of the Constant Register are stored in location 6632)                               |
| *1000;1R                    | (relocation Register 1 is set to 1000)  |
| *1,4272;C= <u>005272</u>    | (relative location 4272 is reprinted as an absolute location and stored in the Constant Register) |

### 8.3.11 Memory Block Initialization, ;F and ;I

The Constant Register can be used in conjunction with the commands ;F and ;I to set a block of memory to a given value. While the most common value required is zero, other possibilities are plus one, minus one, ASCII space, etc.

When the command ;F is typed, ODT stores the contents of the Constant Register in successive memory words starting at the memory word address specified in the lower search limit, and ending with the address specified in the upper search limit.

When the command ;I is typed, the low-order 8 bits in the Constant Register are stored in successive bytes of memory starting at the byte address specified in the lower search limit and ending with the byte address specified in the upper search limit.

For example, assume relocation register 1 contains 7000, 2 contains 10000, and 3 contains 15000. The following sequence sets word locations 7000-7776 to zero, and byte locations 10000-14777 to ASCII spaces.



## On-Line Debugging Technique

```

*$M/000000 <LF>          (open mask register to gain access
nnnnnn /000000 1,0 <LF>  (to search limits)
nnnnnn /000000 2,-2 <LF>  (set lower limit to 7000)
                          (set upper limit to 7776)
*0; C=000000             (Constant Register set to zero)
* ; F                    (Locations 7000-7776 set to zero)
-
*$M/000000 <LF>
nnnnnn /007000 2,0 <LF>   (set lower limit to 10000)
nnnnnn /007776 3,-1 <CR>  (set upper limit to 14777)
*40; C=000040            (Constant Register set to 40
* ; I                    (SPACE))
*                          (Byte locations 10000-14777 are set
                          to value in low-order 8 bits of
                          Constant Register)
```

### 8.3.12 Calculating Offsets, r;0

Relative addressing and branching involve the use of an offset--the number of words or bytes forward or backward from the current location to the effective address. During the debugging session it may be necessary to change a relative address or branch reference by replacing one instruction offset with another. ODT calculates the offsets in response to the r;0 command.

The command r;0 causes ODT to print the 16-bit and 8-bit offsets from the currently open location to address r. For example:

```

*346/000034 414;0 000044 022 22 <CR>
*/000022
```

In the example, location 346 is opened and the offsets from that location to location 414 are calculated and printed. The contents of location 346 are then changed to 22 (the 8-bit offset) and verified on the next line.

The 8-bit offset is printed only if it is in the range -128(decimal) to 127(decimal) and the 16-bit offset is even, as was the case above. For example, the offset of a relative branch is calculated and modified as follows:

```

*1034/103421 1034;0 177776 377 \021 = 377 <CR>
*/103777
```

Note that the modified low-order byte 377 must be combined with the unmodified high-order byte.

### 8.3.13 Relocation Register Commands, r;nR, ;nR, ;R

The use of the relocation registers is defined in Section 8.2. At the beginning of a debugging session it is desirable to preset the registers to the relocation biases of those relocatable modules which will be receiving the most attention.

This can be done by typing the relocation bias, followed by a semicolon and the specification of relocation registers, as follows:



## On-Line Debugging Technique

r;nR

r may be any relocatable expression and n is an integer from 0 to 7. If n is omitted it is assumed to be 0. As an example:

```
*1000;5R      (puts 1000 into relocation register 5)
-5,100;5R     (effectively adds 100 to the contents
-             of relocation register 5)
*
```

In certain uses, programs may be relocated to an address below that at which they were assembled. This could occur with PIC code (Position Independent Code) which is moved without the use of the Linker. In this case the appropriate relocation bias would be the two's complement of the actual downward displacement. One method for easily evaluating the bias and putting it in the relocation register is illustrated in the following example.

Assume a program was assembled at location 5000 and was moved to location 1000. Then the sequence:

```
*1000;1R
-1,-5000;1R
-
*
```

enters the two's complement of 4000 in relocation register 1, as desired.

Relocation registers are initialized to -1, so that unwanted relocation registers never enter into the selection process when ODT searches for the most appropriate register.

To set a relocation register to -1, type ;nR. To set all relocation registers to -1, type ;R.

ODT maintains a table of relocation registers, beginning at the address specified by \$R. Opening \$R (\$R/) opens relocation register 0. Successively typing a line feed opens the other relocation registers in sequence. When a relocation register is opened in this way, it may be modified like any other memory location.

### 8.3.14 The Relocation Calculators, nR and n!

When a location has been opened, it is often desirable to relate the relocated address and the contents of the location back to their relocatable values. To calculate the relocatable address of the opened location relative to a particular relocation bias, type n!, where n specifies the relocation register. This calculator works with opened bytes and words. If n is omitted, the relocation register whose contents are closest but less than or equal to the opened location is selected automatically by ODT. In the following example, assume that these conditions are fulfilled by relocation register 2, which contains 2000. To find the most likely module that a given opened byte is in:

```
*2500\011 = !=2,000500
```

Typing nR after opening a word causes ODT to print the octal number which equals the value of the contents of the opened location minus the contents of relocation register n. If n is omitted, ODT selects



## On-Line Debugging Technique

the relocation register whose contents are closest but less than or equal to the contents of the opened location. For example, assume the relocation bias stored in relocation register 1 is 7000; then:

\*1,500/000000 1R=1,171000

The value 171000 is the content of 1,500, relative to the base 7000. An example of the use of both relocation calculators follows.

If relocation register 1 contains 1000, and relocation register 2 contains 2000, then to calculate the relocatable addresses of location 3000 and its contents, relative to 1000 and 2000, the following can be performed.

\*3000/000410 1!=1,002000 2!=2,001000 1R=1,177410 2R=2,176410

### 8.3.15 ODT Priority Level, \$P

\$P represents a location in ODT that contains the interrupt (or processor) priority level at which ODT operates. If \$P contains the value 377, ODT operates at the priority level of the processor at the time ODT is entered. Otherwise \$P may contain a value between 0 and 7 corresponding to the fixed priority at which ODT operates.

To set ODT to the desired priority level, open \$P. ODT prints the present contents, which may then be changed:

\*\$P/000006 377 <CR>  
\*

If \$P is not specified, its value is seven.

ODT priority must be set to 0 if ODT is being used in an F/B environment with another job running.

Breakpoints may be set in routines which run at different priority levels. For example, a program running at a low priority may use a device service routine which operates at a higher priority level. If a breakpoint occurs from a low-priority routine, ODT operates at a low priority; if an interrupt occurs from a high priority routine, the breakpoints in the high priority routine will not be recognized since they were removed when the low priority breakpoint occurred. That is, interrupts set at a priority higher than the one at which ODT is running will occur and any breakpoints will not be recognized. ODT disables all breakpoints from the program whenever it gains control. Breakpoints are enabled when ;P and ;G commands are executed. For example:

\*\$P/000007 5  
\*1000;B  
\*2000;B  
\*1000;G  
B0;001000  
\* (an interrupt occurs and is serviced)

If a higher level interrupt occurs while ODT is waiting for input the interrupt will be serviced, and no breakpoints will be recognized.



## On-Line Debugging Technique

### 8.3.16 ASCII Input and Output, r;nA

ASCII text may be inspected and changed by the command:

r;nA

where r is a relocatable expression, and n is a character count. If n is omitted it is assumed to be 1. ODT prints n characters starting at location r, followed by a carriage return/line feed. Type one of the following:

<CR> ODT outputs a carriage return/line feed and an asterisk and waits for another command.

<LF> ODT opens the byte following the last byte output.

Up to n characters of text

ODT inserts the text into memory, starting at location r. If fewer than n characters are typed, terminate the command by typing CTRL U, causing a carriage return/line feed/asterisk to be output. However, if exactly n characters are typed, ODT responds with a carriage return/line feed, the address of the next available byte and a carriage return/line feed/asterisk.

ODT does not check the magnitude of n.

## 8.4 PROGRAMMING CONSIDERATIONS

Information in this section is not necessary for the efficient use of ODT. However, it does provide a better understanding of how ODT performs some of its functions and in certain difficult debugging situations, this understanding is necessary.

### 8.4.1 Functional Organization

The internal organization of ODT is almost totally modularized into independent subroutines. The internal structure consists of three major functions: command decoding, command execution, and various utility routines.

The command decoder interprets the individual commands, checks for command errors, saves input parameters for use in command execution, and sends control to the appropriate command execution routine.

The command execution routines take parameters saved by the command decoder and use the utility routines to execute the specified command. Command execution routines exit either to the object program or back to the command decoder.

The utility routines are common routines such as SAVE-RESTORE and I/O. They are used by both the command decoder and the command executors.



## On-Line Debugging Technique

### 8.4.2 Breakpoints

The function of a breakpoint is to give control to ODT whenever the user program tries to execute the instruction at the selected address. Upon encountering a breakpoint, all of the ODT commands can be used to examine and modify the program.

When a breakpoint is executed, ODT removes all the breakpoint instructions from the user's code so that the locations may be examined and/or altered. ODT then types a message on the terminal of the form Bn;k where k is the breakpoint address (and n is the breakpoint number). The breakpoints are automatically restored when execution is resumed.

A major restriction in the use of breakpoints is that the word where a breakpoint was set must not be referenced by the program in any way since ODT altered the word. Also, no breakpoint should be set at the location of any instruction that clears the T-bit. For example:

```
MOV #240,177776      ;SET PRIORITY TO LEVEL 5
```

#### NOTE

Instructions that cause or return from traps (e.g., EMT, RTI) are likely to clear the T-bit, since a new word from the trap vector or the stack is loaded into the Status Register.

A breakpoint occurs when a trace trap instruction (placed in the user program by ODT) is executed. When a breakpoint occurs, the following steps are taken:

1. Set processor priority to seven (automatically set by trap instruction).
2. Save registers and set up stack.
3. If internal T-bit trap flag is set, go to step 13.
4. Remove breakpoints.
5. Reset processor priority to ODT's priority or user's priority.
6. Make sure a breakpoint or single-instruction mode caused the interrupt.
7. If the breakpoint did not cause the interrupt, go to step 15.
8. Decrement repeat count.
9. Go to step 18 if non-zero; otherwise reset count to one.
10. Save terminal status.
11. Type message about the breakpoint or single-instruction mode interrupt.



## On-Line Debugging Technique

12. Go to command decoder.
13. Clear T-bit in stack and internal T-bit flag.
14. Jump to the Go processor.
15. Save terminal status.
16. Type BE (Bad Entry) followed by the address.
17. Clear the T-bit, if set, in the user status and proceed to the command decoder.
18. Go to the Proceed processor, bypassing the TT restore routine.

Note that steps 1-5 inclusive take approximately 100 microseconds during which time interrupts are not permitted (ODT is running at level 7).

When a proceed (;P) command is given, the following occurs:

1. The proceed is checked for legality.
2. The processor priority is set to seven.
3. The T-bit flags (internal and user status) are set.
4. The user registers, status, and Program Counter are restored.
5. Control is returned to the user.
6. When the T-bit trap occurs, steps 1, 2, 3, 13, and 14 of the breakpoint sequence are executed, breakpoints are restored, and program execution resumes normally.

When a breakpoint is placed on an IOT, EMT, TRAP, or any instruction causing a trap, the following occurs:

1. When the breakpoint occurs as described above, ODT is entered.
2. When ;P is typed, the T-bit is set and the IOT, EMT, TRAP, or other trapping instruction is executed.
3. This causes the current PC and status (with the T-bit included) to be pushed on the stack.
4. The new PC and status (no T-bit set) are obtained from the respective trap vector.
5. The whole trap service routine is executed without any breakpoints.
6. When an RTI is executed, the saved PC and PS (including the T-bit) are restored. The instruction following the trap-causing instruction is executed. If this instruction is not another trap-causing instruction, the T-bit trap occurs, causing the breakpoints to be reinserted in the user program,



## On-Line Debugging Technique

or the single-instruction mode repeat count to be decremented. If the following instruction is a trap-causing instruction, this sequence is repeated starting at step 3.

### NOTE

Exit from the trap handler must be via the RTI instruction. Otherwise, the T-bit is lost. ODT can not regain control since the breakpoints have not been reinserted yet.

Note that the ;P command is illegal if a breakpoint has not occurred (ODT responds with ?); ;P is legal, however, after any trace trap entry.

The internal breakpoint status words have the following format:

1. The first eight words contain the breakpoint addresses for breakpoints 0-7. (The ninth word contains the address of the next instruction to be executed in single-instruction mode.)
2. The next eight words contain the respective repeat counts. The following word contains the repeat count for single-instruction mode.)

These words may be changed at will, either by using the breakpoint commands or by direct manipulation with \$B.

When program runaway occurs (that is, when the program is no longer under ODT control, perhaps executing an unexpected part of the program where a breakpoint has not been placed), ODT may be given control by pressing the HALT key to stop the computer, and restarting ODT (see Section 8.1). ODT prints \*, indicating that it is ready to accept a command.

If the program being debugged uses the teleprinter for input or output, the program may interact with ODT to cause an error since ODT uses the teleprinter as well. This interactive error will not occur when the program being debugged is run without ODT.

Note the following rules concerning the ODT break routine:

1. If the teleprinter interrupt is enabled upon entry to the ODT break routine, and no output interrupt is pending when ODT is entered, ODT generates an unexpected interrupt when returning control to the program.
2. If the interrupt of the teleprinter reader (the keyboard) is enabled upon entry to the ODT break routine, and the program is expecting to receive an interrupt to input a character, both the expected interrupt and the character are lost.
3. If the teleprinter reader (keyboard) has just read a character into the reader data buffer when the ODT break routine is entered, the expected character in the reader data buffer is lost.



## On-Line Debugging Technique

### 8.4.3 Searches

The word search allows the user to search for bit patterns in specified sections of memory. Using the \$M/ command, the user specifies a mask, a lower search limit (\$M+2), and an upper search limit (\$M+4). The search object is specified in the search command itself.

The word search compares selected bits (where ones appear in the mask) in the word and search object. If all of the selected bits are equal, the unmasked word is printed.

The search algorithm is:

1. Fetch a word at the current address.
2. XOR (exclusive OR) the word and search object.
3. AND the result of step 2 with the mask.
4. If the result of step 3 is zero, type the address of the unmasked word and its contents. Otherwise, proceed to step 5.
5. Add two to the current address. If the current address is greater than the upper limit, type \* and return to the command decoder, otherwise go to step 1.

Note that if the mask is zero, ODT prints every word between the limits, since a match occurs every time (i.e., the result of step 3 is always zero).

In the effective address search, ODT interprets every word in the search range as an instruction which is interrogated for a possible direct relationship to the search object. The mask register is opened only to gain access to the search limit registers.

The algorithm for the effective address search is (where (X) denotes contents of X, and K denotes the search object):

1. Fetch a word at the current address X.
2. If (X)=K [direct reference], print contents and go to step 5.
3. If (X)+X+2=K [indexed by PC], print contents and go to step 5.
4. If (X) is a relative branch to K, print contents.
5. Add two to the current address. If the current address is greater than the upper limit, perform a carriage return/line feed and return to the command decoder; otherwise, go to step 1.

### 8.4.4 Terminal Interrupt

Upon entering the TT SAVE routine, the following occurs:



## On-Line Debugging Technique

1. Save the LSR status register (TKS).
2. Clear interrupt enable and maintenance bits in the TKS.
3. Save the TT status register (TPS).
4. Clear interrupt enable and maintenance bits in the TPS.

To restore the TT:

1. Wait for completion of any I/O from ODT.
2. Restore the TKS.
3. Restore the TPS.

### NOTES

If the TT printer interrupt is enabled upon entry to the ODT break routine, the following may occur:

1. If no output interrupt is pending when ODT is entered, an additional interrupt always occurs when ODT returns control to the user.
2. If an output interrupt is pending upon entry, the expected interrupt occurs when the user regains control.

If the TT reader (keyboard) is busy or done, the expected character in the reader data buffer is lost.

If the TT reader (keyboard) interrupt is enabled upon entry to the ODT break routine, and a character is pending, the interrupt (as well as the character) is lost.

## 8.5 ERROR DETECTION

ODT detects two types of error: illegal or unrecognizable command and bad breakpoint entry. ODT does not check for the legality of an address when commanded to open a location for examination or modification. Thus the command:

```
*177774/  
?M-TRAP TO 4 003362
```

references nonexistent memory, thereby causing a trap through the vector at location 4. If this vector has not been properly initialized, unpredictable results occur.



## On-Line Debugging Technique

Typing something other than a legal command causes ODT to ignore the command, print:

(echoes illegal command)?  
\*

and wait for another command. Therefore, to cause ODT to ignore a command just typed, type any illegal character (such as 9 or RUBOUT) and the command will be treated as an error, i.e., ignored.

ODT suspends program execution whenever it encounters a breakpoint, i.e., traps to its breakpoint routine. If the breakpoint routine is entered and no known breakpoint caused the entry, ODT prints:

BEEnnnnnn  
\*

and waits for another command. BEEnnnnnn denotes Bad Entry from location nnnnnn. A bad entry may be caused by an illegal trace trap instruction, setting the T-bit in the status register, or by a jump to the middle of ODT.



## CHAPTER 9

### PROGRAMMED REQUESTS

A number of services at the machine language level which the monitor regularly provides to system programs are also available to user-written programs. These include services for file manipulation, command interpretation, and facilities for input and output operations. User programs call these monitor services by means of "programmed requests", which are assembler macro calls written into the user program and interpreted by the monitor at program execution time.

#### NOTE

Programmed requests used in Version 2 differ from those used in Version 1; for example, the channel number in Version 1 was limited to the range 0-17, where it is not in Version 2; blank fields in macro calls were not allowed in Version 1, and are in Version 2; a .area argument points to an argument list in Version 2, where arguments were pushed on the stack in Version 1.

Although programs written for use under Version 1 will assemble and execute properly, it is to the user's advantage to convert these programs so they use the new Version 2 macro calls wherever possible. Only macro calls which are used with the current version of RT-11 (Version 2) are discussed in this chapter. See Section 9.5 for instructions on converting Version 1 macro calls to the Version 2 format.

The macro definitions for both Version 1 and Version 2 requests are included in the file SYSMAC.SML (in 8K systems, the system macro library is called SYSMAC.8K); Appendix D provides a listing of SYSMAC.SML. Refer to Chapter 5 for general information related to the use of macro calls.



## Programmed Requests

### 9.1 FORMAT OF A PROGRAMMED REQUEST

The basis of a programmed request is the EMT instruction, used to communicate information to the monitor. When an EMT is executed, control is passed to the monitor, which extracts appropriate information from the EMT and executes the function required. The low-order byte of the EMT instruction contains a code which is interpreted as:

| <u>Low-Order Byte<br/>of EMT</u> | <u>Meaning</u>  |
|----------------------------------|---|
| 377                              | Reserved; RT-11 ignores this EMT and returns control to the user program immediately.   |
| 376                              | Used internally by the RT-11 monitor; this EMT code should never be used by user programs.  |
| 375                              | Programmed request with several arguments: R0 must point to a list of arguments which designates the specific function.                                       |
| 374                              | Programmed request with one argument: R0 contains a function code in the high-order byte and a channel number (see Section 9.2.1) or 0 in the low-order byte. |
| 360-373                          | Used internally by the RT-11 monitor; these EMT codes should never be used by user programs.  |
| 340-357                          | Programmed request with arguments on the stack and/or in R0.  |
| 0-337                            | Version 1 programmed request. These EMTs use arguments both on the stack and in R0. They are supported for binary compatability with Version 1 programs.      |

A programmed request consists of a macro call followed, where necessary, by one or more arguments. Arguments supplied to a macro call must be legal assembler expressions since arguments will be used as source fields in MOV instructions when the macros are expanded at assembly time. The following two formats are used:

1. PRGREQ ARG1,ARG2,...ARGN
2. PRGREQ AREA,ARG1,ARG2,...ARGN

Form 1 above contains the arguments ARG1 through ARGN; no argument list pointer is required. Macros of this form generate either an EMT 374 or one of the EMTs 340-357. Certain arguments for this form may be omitted, and these are described later in this chapter under the appropriate macro description.

In form 2 above, AREA is a pointer to the argument list which contains the arguments ARG1 through ARGN. This form always causes an EMT 375 to be generated. Blank fields are permitted; however, if the AREA



## Programmed Requests

argument is blank, the monitor assumes that R0 points to a valid argument block (see Section 9.2.3). If any of the fields ARG1 to ARGN are blank, the corresponding entries in the argument list are left untouched. Thus,

```
.PRGREQ AREA,A1,A2
```

points R0 to the argument block at AREA and fills in the first and second arguments, while:

```
.PRGREQ AREA
```

points R0 to the block, and fills in the first word but does not fill in any other arguments. The call:

```
.PRGREQ ,A1
```

assumes R0 points to the argument block and fills in the A1 argument, but leaves the A2 argument alone. The call:

```
.PRGREQ
```

generates only an EMT 375 and assumes that both R0 and the block to which it points are properly set up.

The arguments to RT-11 programmed request macros all serve as the source field of a MOV instruction which moves a value into the argument block or R0. For example:

```
.PRGREQ CHAR
```

expands into:

```
MOV CHAR,R0  
EMT 357
```

Care should be taken to make certain that the arguments specified are legal source fields and that the address accurately represents the value desired. If the value is a constant, immediate mode [#] should be used; if the value is in a register, the register mnemonic [Rn] should be used; if the value is indirectly addressed, the appropriate register convention is necessary [@Rn], and if the value is in memory, the label of the location whose value is the argument is used.

Following are some examples of both correct and incorrect macro calls. Consider the general request:

```
.PRGREQ .AREA,.ARG1,...ARGN
```

A more common way of writing a request of this form is:

```
.PRGREQ #AREA,#ARG1,...#ARGN
```

In this format, the address of AREA is put directly into the argument list. AREA is the tag which indicates the beginning of the argument block. For example:

```
.PRGREQ #AREA,#4  
.  
.  
.  
AREA: .BLKW 3
```



## Programmed Requests

When a direct numerical argument is required, the # causes the correct value to be put into the argument block.. For example:

```
.PRGREQ #AREA,#4
```

is correct, while:

```
.PRGREQ #AREA,4
```

is not. This form interprets the 4 as meaning "move the contents of location 4 into the argument block", where the number 4 itself should be moved into the block.

If the request is written as:

```
.PRGREQ AREA,#4
```

it is interpreted as "use the contents of location AREA as the list pointer", when the address of AREA is actually desired. This expansion could be used with the following form:

```
.PRGREQ LIST1,#4
      .
      .
LIST1: AREA
AREA:  .BLKW3
```

In this case, the content of location LIST1 is the address of the argument list. Similarly, this form is correct:

```
.PRGREQ LIST1,NUMBER
LIST1: AREA
NUMBER: 4
```

In this case, the contents of the locations LIST1 and NUMBER are the argument list pointer and data value, respectively.

### NOTE

All registers except R0 are preserved across a programmed request. (In certain cases, R0 may contain information passed back by the monitor; however, unless the description of a request indicates that a specific value is returned in R0, it may be assumed that the contents of R0 are unpredictable upon return from the request). With the exception of calls to the CSI, the position of the stack pointer is also preserved across a programmed request.



## Programmed Requests

### 9.2 SYSTEM CONCEPTS

Some basic operational characteristics and concepts of RT-11 are described below.

#### 9.2.1 Channel Number (chan)

A channel number is a logical identifier in the range 0 to 377(octal) for a file or "set of data" used by the RT-11 monitor. Thus, when a file is opened on a particular device, a channel number is assigned to that file. To refer to an open file, it is only necessary to refer to the appropriate channel number for that file.

#### 9.2.2 Device block (dblk)

A device block is a four-word block of radix-50 information which specifies a physical device and file name for an RT-11 programmed request. (Refer to Chapter 5 for an explanation of .RAD50 strings.) For example, a device block representing a file FILE.EXT on device DK: could be written as:

```
.RAD50 /DK /  
.RAD50 /FIL/  
.RAD50 /E /  
.RAD50 /EXT/
```

The first word contains the device name, the second and third words contain the file name, and the fourth contains the extension. Device, name, and extension must each be left-justified in the appropriate field. This string could also be written as:

```
.RAD50 /DK FILE EXT/
```

Note that spaces must be used to fill out each field. Note also that the colon and period separators do not appear in the actual RAD50 string. They are used only by the monitor keyboard interface to delimit the various fields.

#### 9.2.3 EMT Argument Blocks

Programmed requests which call the monitor via EMT 375 use R0 as a pointer to an argument list. In general, this argument list appears as follows:

| <u>address</u> | <u>contents</u>  |
|----------------|------------------|
|                | Function Channel |
| x              | Code Number      |
| x+2            | argument1        |
| x+4            | argument2        |
| .              | .                |
| .              | .                |
| .              | .                |



## Programmed Requests

R0 points to location x. The even (low-order) byte of location x contains the channel number named in the macro call. If no channel number is required, the byte is set to 0. The odd (high-order) byte of x is a code specifying the function to be performed. Locations x+2, x+4, etc. contain arguments to be interpreted. These are described in detail under each request.

Requests which use EMT 374 set up R0 with the channel number in the even byte and the function code in the odd byte. They require no other arguments.

### 9.2.4 Important Memory Areas

9.2.4.1 Vector Addresses (0-37, 60-477) - Certain areas of memory between 0 and 477 are reserved for use by RT-11. KMON does not load these locations from the save image file when it initiates a program, i.e., R, RUN, and GET will not load these words. However, no hardware memory protection is supplied. Thus, programs should never alter the contents of the indicated areas at run-time.

| <u>Locations</u> | <u>Contents</u>  |
|------------------|--|
| 0,2              | Monitor restart. Executes .EXIT request and returns control to KMON.               |
| 4,6              | Time out or bus error trap; RT-11 sets this to point to its internal trap handler. |
| 10,12            | Reserved instruction trap; RT-11 sets this to point to its internal trap handler.  |
| 30,32            | EMT trap vector and status.  |
| 40-57            | RT-11 system communication area (see below).                                       |
| 60,62            | TTY input interrupt vector and status.   |
| 64,66            | TTY output interrupt vector and status.  |
| 100,102          | KW11L vector and status.   |
| 204,206          | RF11 vector and status.  |
| 214,216          | TC11 vector and status.  |
| 220,222          | RK05 vector and status.  |
| 330,332          | GT40 shift out interrupt vector and status.  |

These areas are not replaced by RT-11. If they are destroyed by a program, the system must be re-bootstrapped, or the program must restore them.



## Programmed Requests

9.2.4.2 Resident Monitor - Section 2.4 of Chapter 2 describes the placement of monitor components when either the Single-Job Monitor or F/B Monitor is brought into memory; included is the approximate size of each monitor component and the size of the area available for handlers and user programs.

9.2.4.3 System Communication Area - RT-11 uses bytes 40-57 to hold information about the program currently executing, as well as certain information used only by the monitor. A description of these bytes follows:

| <u>Bytes</u> | <u>Meaning and Use</u>   |
|--------------|--|
| 40,41        | Start address of job. When a file is linked into an RT-11 memory image, this word is set to the starting address of the job either with the Linker /T switch or as an argument in the .END statement of the program. When a foreground program is executed, the FRUN processor relocates this word to contain the actual starting address of the program.  |
| 42,43        | Initial value of the stack pointer. If it is not set by the user program in an .ASECT, it defaults to 1000 or the top of the .ASECT in the background, whichever is larger. If a foreground program does not specify a stack pointer in this word, a default stack (128 decimal words) is allocated by FRUN immediately below the program. The initial stack pointer can also be set with the Linker /M switch option. |
| 44,45        | Job Status Word. Used as a flag word for the monitor. Certain bits are maintained by the monitor exclusively while others must be set or cleared by the user job. Those bits in the following list which are marked by an asterisk are bits which must be set by the user job.   |

Since the currently unassigned bits may be used in future releases of RT-11, user programs should not use these bits for internal flags.

| <u>Bit Number</u> | <u>Meaning</u>   |
|-------------------|--|
| 15                | USR swap bit. (Unused in F/B.) Programs which do not require the USR to be swapped have this bit set. See the Swapping Algorithm (Section 9.2.5) for more details. |
| 14                | Unused.  |
| *13               | Reenter bit. When set, this bit indicates that the program may be restarted from the terminal with the REENTER command.  |



## Programmed Requests

- \*12 Special mode TT bit. When set, this bit indicates that the job is in a "special" keyboard mode of input. Refer to the explanation of the .TTYIN/.TTINR requests for details.
- 11-10 For F/B Monitor use only.
- 9 Overlay Bit. Set (by the Linker) if the job uses the Linker overlay structure.
- 8 CHAIN bit. If this bit is set in a job's save image, words 500-776 are loaded from the save file when the job is started even if the job is entered via CHAIN. (These words are normally used to pass parameters across CHAINs.) The bit is set when a job is running if and only if the job was actually entered with CHAIN.
- \*7 Error halt bit. When set, this bit indicates a halt on an I/O error. If the user desires to halt when any I/O device error occurs, this bit should be set. (Unused in F/B.)
- \*6 Inhibit TT wait bit. For use with the Foreground/Background system. When set, this bit inhibits the monitor from entering a console terminal wait state. Refer to the sections concerning .TTYIN/.TTINR, and .TTYOUT/.TTOUTR for more information.
- 5-0 Unused.
- 46,47 USR load address. Normally 0, this word may be set to any valid word address in the user's program. See Section 9.2.5, Swapping Algorithm, for details of use.
- 50,51 High memory address. The monitor maintains the highest address the user program can use in this word. The Linker sets it initially. It is modified only via the .SETTOP (Set Top of Memory) monitor request.
- 52 EMT error code. If a monitor request results in an error, the code number of the error is always returned in byte 52. Each monitor call has its own set of possible errors. It is recommended that the user program reference byte 52 with absolute addressing, rather than relative addressing. For example:

```
ERRWRD = 52
TSTB ERRWRD           ;RELATIVE ADDRESSING
TSTB @#ERRWRD         ;ABSOLUTE ADDRESSING
```



## Programmed Requests

Location 52 should always be addressed as a byte, never as a word, since byte 53 may be used in future releases of RT-11.

- 53           Reserved for future system use.
- 54,55       Address of the beginning of the Resident Monitor. RT-11 always loads the resident into the highest available memory locations; this word points to its first location. It must never be altered by the user. Doing so will cause RT-11 to malfunction.
- 56           Fill character (7-bit ASCII). Some high-speed terminals require filler (null) characters after printing certain characters. Byte 56 should contain the ASCII 7-bit representation of the character after which fillers are required.
- 57           Fill count. This byte specifies the number of fill characters required. If bytes 56 and 57=0, no fillers are required.

The required fill characters are:

| <u>Terminal</u>        | <u>No. of fills</u>      | <u>Value of Word 56</u> |
|------------------------|--------------------------|-------------------------|
| Serial LA30 @ 300 baud | 10 after carriage return | 5015                    |
| Serial LA30 @ 150 baud | 4 after carriage return  | 2015                    |
| Serial LA30 @ 110 baud | 2 after carriage return  | 1015                    |
| VT05 @ 2400 baud       | 4 after line feed        | 2012                    |
| VT05 @ 1200 baud       | 2 after line feed        | 1012                    |
| VT05 @ 600 baud        | 1 after line feed        | 412                     |

### 9.2.5 Swapping Algorithm

Programmed requests are divided into two categories according to whether or not they require the USR to be in memory (see Table 9-2). Any request which requires the USR in memory may also require that a portion of the user program be saved temporarily on the system device scratch blocks (i.e., be "swapped out") to provide room for the USR. The USR will be read into the swapped region.

During most normal operations, this swapping is invisible to the user and he need not be concerned about it. However, it is possible to optimize programs so that they require little or no swapping. This is particularly useful when operating in an F/B environment, since under the F/B system, the USR will be swapped for both background and foreground jobs regardless of which job required it. If the USR is not swapped, neither the foreground nor the background job will be slowed down by the swapping process.

The following items should be considered if a swap operation is necessary:

1. The background job - If a .SETTOP request in a background job specifies an address beyond the point at which the USR



## Programmed Requests

normally resides, a swap will be required when the USR is called. More details concerning the .SETTOP request are in Section 9.4.36.

2. The value of location 46 - If the user either assembles an address into word 46 or moves a value there while the program is running, RT-11 uses the contents of that word as an alternate place to swap the USR. If location 46 is 0, this indicates that the USR will be at its normal location in high memory.

### NOTES

1. If the USR does not require swapping, the value in location 46 is ignored. Swapping is a relatively time-consuming operation and is avoided, if possible.
2. A foreground job should always have a value in location 46 unless it is certain that the USR will never be swapped. If the foreground job does not allow space for the USR and a swap is required, a fatal error occurs. (The SET USR NOSWAP command, explained in Chapter 2, ensures that the USR will be resident.)
3. Care should be taken when specifying an alternate address to location 46. The single-job system does not verify the legality of the USR swap address. Thus, if the area to be swapped overlays the Resident Monitor, the system is destroyed.
4. The user should also take care that the USR is never swapped over any of the following areas: the program stack; any parameter block for calls to the USR; any I/O buffers, device handlers, or completion routines being used when the USR is called.

The following is an example of the way a background program can use all memory up to, but not including, the USR. This leaves the USR resident and therefore does not require swapping.

```
.MCALL ..V2...REGDEF,.SETTOP,.EXIT
..V2..
.REGDEF

RMPTR=54          ;POINTER TO RMON IS AT 54.
USRLOC=266        ;POINTER TO USR LOCATION IS
                  ;AT 266 BYTES INTO RMON.

START:
MOV      0,RMPTR,R1      ;R1 -> RESIDENT MONITOR
```



## Programmed Requests

```

MOV      USRLOC(R1),R1    /R1 => USR
.SETTOP  R1              /ASK FOR MEMORY UP TO USR
MOV      R0,HILIM        /R0 = HIGH LIMIT OF MEMORY
                          /ACTUALLY GRANTED BY MONITOR.

HILIM:   .EXIT
          .WORD 0         /CONTAINS HI LIMIT OF MEMORY
          .END  START

```

### 9.2.6 Offset Words

There are several words which always have fixed positions relative to the start of the Resident Monitor. It is often advantageous for user programs to be able to access these words. This is done with the code:

```

RMON = 54
MOV @#RMON,register
MOV OFFSET(register),register

```

Here, register is any general register and OFFSET is a number from the following list:

| <u>OFFSET (Bytes)</u> | <u>Contents</u>  |
|-----------------------|--|
| 262                   | System date. (See .DATE request.)  |
| 266                   | Start of normal USR area. This is where the USR will reside when it is non-swapping. It is useful to be able to perform a .SETTOP in a background job such that the USR is always resident. (An example is in Section 9.2.5.)                    |
| 270                   | Address of I/O exit routine for all devices. The exit routine is an internal queue management routine through which all device handlers exit once the I/O transfer is complete. Any new devices added to RT-11 must also use this exit location. |
| 276                   | Monitor version number (2-377). The user can always access the version number to determine if the most recent monitor is in use.   |
| 277                   | Update number. Patches to the monitor always increment the update number. This provides a means of checking that all patches have been made. (This number should be accessed by MOVb rather than MOV).   |
| 300                   | Configuration word. This is a string of 16 bits used to indicate information about either the hardware configuration of the system, or a software condition. The bits and their meanings are:  |



## Programmed Requests

| <u>Bit #</u> | <u>Meaning</u>   |
|--------------|--|
| 0            | 0 = Single-Job Monitor<br>1 = F/B Monitor  |
| 2            | 1 = GT40 display hardware exists   |
| 3            | 1 = RT-11 BATCH is in control<br>of the background (note that<br>the BATCH processor is not<br>available at this time) |
| 5            | 0 = 60-cycle clock<br>1 = 50-cycle clock   |
| 6            | 1 = 11/45 floating-point<br>hardware exists  |
| 7            | 0 = No foreground job is in memory<br>1 = Foreground job is in memory  |
| 8            | 1 = User is linked to the GT40<br>scroller   |
| 9            | 1 = USR is permanently resident<br>(via a SET USR NOSWAP)  |
| 15           | 1 = KW11L clock is present   |

The other bits are reserved for future use and should not be accessed by user programs.

304-313

These locations contain the addresses of the console terminal control and status registers. The order is:

304 Keyboard status  
306 Keyboard buffer  
310 Printer status  
312 Printer buffer

These locations can be changed, for example, to reflect a second terminal; thus RT-11 can be made to run on any terminal present on the system which is connected to the machine via the DL11 multiple terminal interface. (Refer to the RT-11 SOFTWARE SUPPORT MANUAL (DEC-11-ORPGA-B-D)).

314

The maximum file size allowed in a 0 length .ENTER. This can be adjusted by the user program or by using the PATCH program to be any reasonable value. The default value is 177777 (decimal) blocks, allowing an essentially unlimited file size.

324

Address of .SYNCH entry. User interrupt routines may enter the monitor through this address to synchronize with the job they are servicing.

354

Address of VT-11 display processor display stop interrupt vector.



## Programmed Requests

### 9.2.7 File Structure

RT-11 uses a "contiguous" file structure. This type of structure implies that every file on the device is made up of a contiguous group of physical blocks. Thus, a file that is 9 blocks long occupies 9 contiguous blocks on the device.

A contiguous area on a device can be in one of the following categories:

1. Permanent file. This is a file which has been .CLOSED on a device. Any named files which appear in a PIP directory listing are permanent files.
2. Tentative file. Any file which has been created via .ENTER, but not .CLOSED, is a tentative file entry. When the .CLOSE request is given, the tentative entry becomes a permanent file. If a permanent file already exists under the same name, the old file is deleted. If a .CLOSE is never given, the tentative file is treated like an empty entry.
3. Empty entry. When disk space is unused or a permanent file is deleted, an empty entry is created. Empty entries appear in a PIP /E directory listing as <UNUSED> N, where N is the decimal length of the empty area.

Since a contiguous structure does not automatically reclaim unused disk space, the device may eventually become "fragmented". A device is fragmented when there are many empty entries which are scattered over the device. RT-11 PIP has an option which allows the user to collect all empty areas so that they occur at the end of a device. Refer to Chapter 4 for details.

### 9.2.8 Completion Routines

Completion routines are user-written routines which are entered following an operation. On entry to a completion routine, R0 contains the channel status word for the operation; R1 contains the octal channel number of the operation. The carry bit is not significant.

Completion routines are handled differently in the Single-Job and the F/B versions of RT-11. In the Single-Job version, completion routines are totally asynchronous and can interrupt one another. In F/B, completion routines do not interrupt each other. Instead they are queued and made to wait until the correct job is running. For example, if a foreground job is running and an I/O transfer initiated by a background job completes and wants to go to a completion routine, the background routine is queued and will not execute until the foreground gives up control of the system. If the foreground is running and a foreground I/O transfer completes and wants a completion routine, that routine will be entered immediately if the foreground is not already inside a completion routine. If it is in a completion routine, that routine continues to termination, at which point any other completion routines are entered in a first in/first out manner. If the background is running and a foreground I/O transfer completes and needs a completion routine, the background is suspended and the foreground routine is entered immediately.



## Programmed Requests

The restrictions which must be observed when writing completion routines are:

1. Completion functions cannot issue a request which would cause the USR to be swapped in. They are primarily used for issuing READ/WRITE commands, not for opening or closing files, etc. A fatal monitor error is generated if the USR is called from a completion routine.
2. Completion routines should never reside in the memory space which will be used for the USR, since the USR can be interrupted when I/O terminates and the completion routine is entered. If the USR has overlaid the routine, control passes to a random place in the USR, with a HALT or error trap the likely result.
3. The routine must be exited via an RTS PC, as it is called from the monitor via a JSR PC,ADDR where ADDR is the user-supplied address.
4. If a completion routine uses registers other than R0 or R1, it must save them upon entry and restore them before exiting.

### 9.2.9 Using the System Macro Library

User programs for RT-11 should always be written using the system macro library (SYSMAC.SML), supplied with RT-11. This ensures compatibility among all user programs and allows easy modification by redefining a macro. A listing of SYSMAC.SML appears in Appendix D.

The system macro library for 8K systems appears on the system device as SYSMAC.8K.

Suggestions for writing foreground programs are in Appendix H, F/B Programming and Device Handlers. This appendix should be read in conjunction with Chapter 9 before coding F/B programs.

## 9.3 TYPES OF PROGRAMMED REQUESTS

There are three types of services which the monitor makes available to the user through programmed requests. These are:

1. Requests for File Manipulation
2. Requests for Data Transfer
3. Requests for Miscellaneous Services

Table 9-1 summarizes the programmed requests in each of these categories alphabetically. Those marked with an asterisk function only in a F/B environment; they are ignored under the Single-Job Monitor. The EMT and function code for each request (where applicable) are included.



# Programmed Requests

Table 9-1  
Summary of Programmed Requests

| Mnemonic                     | EMT & Code |    | Section | Purpose  |
|------------------------------|------------|----|---------|--|
| File Manipulation Requests   |            |    |         |  |
| *.CHCOPY                     | 375        | 13 | 9.4.3   | Establishes a link and allows one job to access another job's channel.   |
| .CLOSE                       | 375        | 7  | 9.4.4   | Closes the specified channel.  |
| .DELETE                      | 375        | 0  | 9.4.10  | Deletes the file from the specified device.  |
| .ENTER                       | 375        | 2  | 9.4.13  | Creates a new file for output.   |
| .LOOKUP                      | 375        | 1  | 9.4.21  | Opens an existing file for input and/or output via the specified channel.  |
| .RENAME                      | 375        | 4  | 9.4.32  | Changes the name of the indicated file to a new name.  |
| .REOPEN                      | 375        | 6  | 9.3.33  | Restores the parameters stored via a SAVESTATUS request and reopens the channel for I/O.   |
| .SAVESTATUS                  | 375        | 5  | 9.4.34  | Saves the status parameters of an open file in user memory and frees the channel for future use.   |
| Data Transfer Requests       |            |    |         |  |
| *.RCVD<br>*.RCVDW<br>*.RCVDC | 375        | 26 | 9.4.29  | Receives data. Allows a job to read messages or data sent by another job in an F/B environment. The three modes correspond to the READ, .READC, and READW modes.   |
| .READ                        | 375        | 10 | 9.4.30  | Transfers data via the specified channel to a memory buffer and returns control to the user program when the transfer request is entered in the I/O queue. No special action is taken upon completion of I/O.  |
| .READC                       | 375        | 10 | 9.4.30  | Transfers data via the specified channel to a memory buffer and returns control to the user program when the transfer request is entered in the I/O queue. Upon completion of the read, control transfers asynchronously to the routine specified in the .READC request. |



| Mnemonic                     | EMT & Code |    | Section | Purpose  |
|------------------------------|------------|----|---------|--|
| .READW                       | 375        | 10 | 9.4.30  | Transfers data via the specified channel to a memory buffer and returns control to the user program only after the transfer is complete.   |
| *.SDAT<br>*.SDATC<br>*.SDATW | 375        | 25 | 9.4.35  | Allows the user to send messages or data to the other job in an F/B environment. The three modes correspond to the .WRITE, .WRITC and .WRITW modes.  |
| .TTYIN<br>.TTINR             | 340        | -- | 9.4.43  | Transfers one character from the keyboard buffer to R0.  |
| .TTYOUT<br>.TTOUTR           | 341        | -- | 9.4.44  | Transfers one character from R0 to the terminal input buffer.  |
| .WRITE                       | 375        | 11 | 9.4.47  | Transfers data via the specified channel to a device and returns control to the user program when the transfer request is entered in the I/O queue. No special action is taken upon completion of the I/O.   |
| .WRITC                       | 375        | 11 | 9.4.47  | Transfers data via the specified channel to a device and returns control to the user program when the transfer request is entered in the I/O queue. Upon completion of the write, control transfers asynchronously to the routine specified in the .WRITC request. |
| .WRITW                       | 375        | 11 | 9.4.47  | Transfers data via the specified channel to a device and returns control to the user program only after the transfer is complete.  |
| Miscellaneous Services       |            |    |         |  |
| .CDFN                        | 375        | 15 | 9.4.1   | Defines additional channels for doing I/O.   |
| .CHAIN                       | 374        | 10 | 9.4.2   | Chains to another program (in the background job only).  |
| *.CMKT                       | 375        | 23 | 9.4.5   | Cancels an unexpired mark time request.  |
| *.CNTXSW                     | 375        | 33 | 9.4.6   | Requests that the indicated memory locations be part of the F/B context switch process.  |
| .CSIGEN                      | 344        | -- | 9.4.7   | Calls the Command String Interpreter (CSI) in general mode.  |
| .CSISPC                      | 345        | -- | 9.4.8   | Calls the CSI in special mode.   |



| Mnemonic  | EMT & Code |    | Section | Purpose  |
|-----------|------------|----|---------|--|
| *.CSTAT   | 375        | 27 | 9.4.9   | Returns the status of the channel indicated.   |
| .DATE     | ---        | -- | 9.3.1.1 | Moves the current date information into R0.  |
| *.DEVICE  | 375        | 14 | 9.4.11  | Allows user to turn off device interrupt enable in F/B upon program termination.   |
| .DSTATUS  | 342        | -- | 9.4.12  | Returns the status of a particular device.   |
| .EXIT     | 350        | -- | 9.4.14  | Exits the user program and returns control to the Keyboard Monitor.  |
| .FETCH    | 343        | -- | 9.4.15  | Loads device handlers into memory.   |
| .GTIM     | 375        | 21 | 9.4.16  | Gets time of day.  |
| .GTJB     | 375        | 20 | 9.4.17  | Gets parameters of this job.   |
| .HERR     | 374        | 5  | 9.4.18  | Specifies termination of the job on fatal errors.  |
| .HRESET   | 357        | -- | 9.4.19  | Terminates I/O transfers and does a .SRESET operation.   |
| .INTEN    | ---        | -- | 9.3.1.2 | Notifies monitor that an interrupt has occurred and to switch to "system state", and sets the processor priority to the correct value.             |
| .LOCK     | 346        | -- | 9.4.20  | Makes the monitor User Service Routines (USR) permanently resident until.EXIT or.UNLOCK is executed. The user program is swapped out if necessary. |
| *.MRKT    | 375        | 22 | 9.4.22  | Marks time; i.e., sets asynchronous routine to occur after a specified interval.   |
| *.MWAIT   | 374        | 11 | 9.3.23  | Waits for messages to be processed.  |
| .PRINT    | 351        | -- | 9.4.24  | Outputs an ASCII string to the terminal.   |
| *.PROTECT | 375        | 31 | 9.4.25  | Requests that vectors in the area from 0-476 be given exclusively to this job.   |
| .PURGE    | 374        | 3  | 9.4.26  | Clears out a channel.  |
| .QSET     | 353        | -- | 9.4.27  | Expands the size of the monitor I/O queue.   |
| .RCTRLO   | 355        | -- | 9.4.28  | Enables output to the terminal.  |

(continued on next page)



| Mnemonic | EMT & Code |    | Section | Purpose   |
|----------|------------|----|---------|---|
| .REGDEF  | ---        | -- | 9.3.1.3 | Defines the PDP-11 general registers.   |
| .RELEAS  | 343        | -- | 9.4.31  | Removes device handlers from memory.  |
| *.RSUM   | 374        | 2  | 9.4.39  | Causes the main line of the job to be resumed where it was suspended with .SPND.                      |
| .SERR    | 374        | 4  | 9.4.18  | Inhibits most fatal errors from causing the job to be aborted.  |
| .SETTOP  | 354        | -- | 9.4.36  | Specifies the highest memory location to be used by the user program.                                 |
| .SFPA    | 375        | 30 | 9.4.37  | Sets user interrupt for floating point processor exceptions.  |
| .SPFUN   | 375        | 32 | 9.4.38  | Performs special functions on magtape and cassette units.   |
| *.SPND   | 374        | 1  | 9.4.39  | Causes the running job to be suspended.   |
| .SRESET  | 352        | -- | 9.4.40  | Resets all channels and releases the device handlers from memory.                                     |
| .SYNCH   | ---        | -- | 9.3.1.4 | Enables user program to perform monitor programmed requests from within an interrupt service routine. |
| *.TLOCK  | 374        | 7  | 9.4.41  | Indicates if the USR is currently being used by another job and performs a .LOCK if available.        |
| .TRPSET  | 375        | 3  | 9.4.42  | Sets a user intercept for traps to locations 4 and 10.  |
| *.TWAIT  | 375        | 24 | 9.4.45  | Suspends the running job for a specified amount of time.  |
| .UNLOCK  | 347        | -- | 9.4.20  | Releases USR if a LOCK was done. The user program is swapped in if required.                          |
| ..V2..   | ---        | -- | 9.3.1.5 | Enables expansions to occur in Version 2 format.  |
| .WAIT    | 374        | 0  | 9.4.46  | Waits for completion of all I/O on a specified channel.   |

Requests requiring the USR (as explained in Section 9.2.5) differ between the Single-Job and F/B Monitors. Table 9-2 indicates which requests require the USR to be in memory. Those requests marked by an asterisk are Version 2 macros only. The CLOSE request on non-file structured devices (LP, PP, TT, etc.) does not require the USR under either monitor.



# Programmed Requests

Table 9-2  
Requests Requiring the USR

| Request              | F/B | Single-Job |
|----------------------|-----|------------|
| *.CDFN               | No  | Yes        |
| *.CHAIN              | No  | No         |
| *.CHCOPY             | No  | N/A        |
| .CLOSE (see Note 1)  | Yes | Yes        |
| *.CMKT               | No  | N/A        |
| *.CNTXSW             | No  | N/A        |
| .CSIGEN              | Yes | Yes        |
| .CSISPC              | Yes | Yes        |
| *.CSTAT              | No  | N/A        |
| .DELETE              | Yes | Yes        |
| *.DEVICE             | No  | N/A        |
| .DSTATUS             | Yes | Yes        |
| .ENTER               | Yes | Yes        |
| .EXIT                | No  | No         |
| .FETCH               | Yes | Yes        |
| *.GTIM               | No  | No         |
| *.GTJB               | No  | No         |
| *.HERR               | No  | No         |
| .HRESET              | No  | Yes        |
| .LOCK (see Note 2)   | Yes | Yes        |
| .LOOKUP              | Yes | Yes        |
| *.MRKT               | No  | N/A        |
| *.MWAIT              | No  | N/A        |
| .PRINT               | No  | No         |
| *.PROTECT            | No  | N/A        |
| *.PURGE              | No  | No         |
| .QSET                | Yes | Yes        |
| .RCTRLO              | No  | No         |
| *.RCVD/RCVDC/RCVDW   | No  | N/A        |
| .READ/READC/READW    | No  | No         |
| .RELEAS              | Yes | Yes        |
| .RENAME              | Yes | Yes        |
| .REOPEN              | No  | No         |
| *.RSUM               | No  | N/A        |
| .SAVESTATUS          | No  | No         |
| *.SDAT/SDATC/SDATW   | No  | N/A        |
| *.SERR               | No  | No         |
| .SETTOP              | No  | No         |
| *.SFPA               | No  | No         |
| *.SPFUN              | No  | No         |
| *.SPND               | No  | N/A        |
| *.SRESET             | No  | Yes        |
| *.TLOCK (see Note 3) | No  | No         |
| *.TRPSET             | No  | No         |
| .TTINR/TTYIN         | No  | No         |
| .TTOUTR/.TTYOUT      | No  | No         |
| *.TWAIT              | No  | N/A        |
| .UNLOCK              | No  | No         |
| .WAIT                | No  | No         |
| .WRITE/WRITC/WRITW   | No  | No         |



## Programmed Requests

Note 1: Only if channel was opened via .ENTER.

Note 2: Only if USR is in a swapping state.

Note 3: Only if USR is not in use by the other job.

### 9.3.1 System Macros

The following five macros are included in the system macro library, but are not programmed requests in that they cause no EMT instruction to be generated:

|         |        |
|---------|--------|
| .DATE   | .SYNCH |
| .INTEN  | ..V2.. |
| .REGDEF |        |

They can be used in the same manner as the other macro calls; their explanations follow.

.DATE

#### 9.3.1.1 .DATE

This request moves the current date information from the system date word into R0. The date word returned is in the following format:

Bit:           14       10 9           5 4           0

|  |                  |                |                   |
|--|------------------|----------------|-------------------|
|  | MONTH<br>(1-12.) | DAY<br>(1-31.) | YEAR-72 (DECIMAL) |
|--|------------------|----------------|-------------------|

Macro Call:           .DATE

Errors:

No errors are returned. A zero result in R0 indicates that no DATE command was entered.



## Programmed Requests

**.INTEN**

### 9.3.1.2 .INTEN

This request is used by user program interrupt service routines to:

1. Notify the monitor that an interrupt has occurred and to switch to "system state",
2. Set the processor priority to the correct value.

In Version 2 of RT-11, all external interrupts cause the processor to go to level 7 (see Appendix H). .INTEN is used to lower the priority to the value at which the device should be run. On return from .INTEN, the device interrupt can be serviced, at which point the interrupt routine returns via an RTS PC. It is very important to note that an RTI will not return correctly from an interrupt routine which specifies an .INTEN.

Macro Call: .INTEN .priority, pic

where: .priority is the processor priority at which the user wishes to run his interrupt routine.

pic is an optional argument which should be non-blank if the interrupt routine is written as a PIC (position independent code) routine. If the routine does not have to be PIC, it is recommended that the PIC field be left blank; the non-PIC version is slightly faster than the PIC version.

The user is advised to read Appendix H for more details concerning the use of .INTEN and .SYNCH.

Errors:

None.

Example:

Refer to Section 9.3.1.4, .SYNCH, for an example.



## Programmed Requests

### .REGDEF

#### 9.3.1.3 .REGDEF

This macro call defines the PDP-11 general registers as R0 through R5, SP, and PC.

Macro Call:   .MCALL .REGDEF,...  
              .REGDEF

Errors:

None.

Example:

Refer to the example for the .SYNCH request. Appendix D shows the expansion of .REGDEF.

### .SYNCH

#### 9.3.1.4 .SYNCH

This macro call enables the user program to perform monitor programmed requests from within an interrupt service routine. Unless a .SYNCH is used, issuing programmed requests from interrupt routines is not supported by the system and should not be performed. .SYNCH, like .INTEN and .DATE, is not a programmed request and generates no EMT instructions.

Macro Call: .SYNCH .area

where:    .area    is the address of a seven-word area which the user must set aside for use by .SYNCH. The 7-word block appears as:

- Word 1   RT-11 maintains this word; its contents should not be altered by the user.
- Word 2   The current job's number. This can be obtained by a .GTJB call.
- Word 3   Unused.



## Programmed Requests

Word 4 Unused.  
Word 5 R0 argument. When a successful return is made from .SYNCH, R0 contains this argument.  
Word 6 Must be -1.  
Word 7 Must be 0.

### Note:

.SYNCH assumes that the user has not pushed anything on the stack between the .INTEN and .SYNCH calls. This rule must be observed for proper operation.

### Errors:

The monitor returns to the location immediately following the .SYNCH if the .SYNCH was rejected. The routine is still unable to issue programmed requests, and R4 and R5 are available for use. Errors returned are due to one of the following:

1. Another .SYNCH which specified the same 7-word block is still pending.
2. An illegal job number was specified in the second word of the block. The only currently legal job numbers are 0 and 2.
3. If the job has been aborted or for some reason is no longer running, the .SYNCH will fail.

Normal return is to the word after the error return with the routine in user state and thus allowed to issue programmed requests. R0 contains the argument which was in word 4 of the block. R0 and R1 are free to be used without having to be saved. (R4 and R5 are not free.) Exit from the routine should be done via an RTS PC.

### Example:

```
.MCALL ..V2...REGDEF
..V2..
.REGDEF
START: .MCALL .GTJB,.INTEN,.WRITC,.SYNCH,.EXIT,.PRINT
      MOV     #JOB,R5          ;OUTPUT OF .GTJB GOES HERE
      .GTJB   #AREA,R5         ;GET JOB NUMBER
      MOV     (R5),SYNBLK+2    ;STORE THE JOB NUMBER INTO SYNCH BLOCK
                                   ;IN HERE WE SET UP INTERRUPT
                                   ;PROCESSING, AND START UP THE
                                   ;INTERRUPTING DEVICE.

INTRPT: .INTEN 5               ;GO INTO SYSTEM STATE
                                   ;RUN AT LEVEL FIVE
                                   ;INTERRUPT PROCESSING --
                                   ;NOTHING CAN GO ON STACK
      .SYNCH   #SYNBLK         ;TIME TO WRITE A BUFFER
      BR       SYNFAIL        ;SYNCH FAILED
                                   ;SYNCH SUCCEEDED
      .WRITC   #AREA,CHAN,BUFF,WCNT,#CRTN1,BLK
                                   ;WRITE A BUFFER
```



## Programmed Requests

|          | BCS          | WTFAIL | IFAILED SOMEHOW              |
|----------|--------------|--------|------------------------------|
|          | RTS          | PC     | IRE-INITIALIZE FOR MORE      |
| SYNBLK1  | .WORD 0      | 0      | INTERRUPTS AND EXIT          |
|          | .WORD 0      | 0      | IJOB NUMBER                  |
|          | .WORD 0      | 0      |                              |
|          | .WORD 5      | 5      | IR0 CONTAINS 5 ON SUCCESSFUL |
|          |              |        | I.SYNCH                      |
| SYNFAIL1 | .WORD 0,-1,0 |        | SET UP FOR MONITOR           |
|          | .            |        |                              |
|          | .            |        |                              |
|          | .            |        |                              |

**..V2..**

### 9.3.1.5 ..V2..

This macro enables macro expansions to occur in Version 2 format. If ..V2.. is not used, all macro expansions will be in Version 1 format. Note that any requests marked with an asterisk in Table 9-1 are not valid as Version 1 requests, and thus will be flagged as errors when they are assembled in Version 1 form. If ..V2.. is used, all macros are generated in Version 2 form. Note also that using ..V2.. causes a symbol ...V2 to be defined. User programs should not use this symbol.

Macro Call: .MCALL ..V2..  
 ..V2..

..V2.. should first be loaded with an .MCALL command and then invoked.

#### Note:

It is possible that user programs will exist in which both Version 1 and Version 2 macros are present. To allow proper assembly, the user should not call ..V2.. as described; instead he should include the statement:

```
.MCALL ...CM1,...CM2,...CM3,...CM4
```

to define the utility macros (CM1, CM2, etc.) used by other Version 2 macros. This causes all macros which existed in Version 1 to assemble in Version 1 format, while all macros which are new to Version 2 are correctly generated as Version 2 macros.

#### Example:

Refer to the example for the .CDFN directive.



## Programmed Requests

### 9.4 PROGRAMMED REQUEST USAGE

This section provides a description of each of the programmed requests alphabetically. The following parameters are commonly used as arguments in the various calls:

|        |  |
|--------|--|
| .addr  | an address, the meaning of which depends on the request being used   |
| .area  | a pointer to the EMT argument list (for those requests which require a list); see Section 9.2.3                  |
| .blk   | a block number specifying the relative block in a file where an I/O transfer is to begin                         |
| .buff  | a buffer address specifying a memory location into or from which an I/O transfer is to be performed              |
| .chan  | a channel number in the range 0-377(octal)   |
| .crtn  | the entry point of a completion routine; see Section 9.2.8   |
| .count | file number for magtape/cassette operations (see Appendix H); if this argument is blank, a value of 0 is assumed |
| .dblk  | the address of a four-word RAD50 descriptor of the file to be operated upon; see Section 9.2.2                   |
| .num   | a number, the value of which depends on the request  |
| .wcnt  | a word count specifying the number of words to be transferred to or from the buffer during an I/O operation      |

Additional information concerning these parameters (and others not defined here) is provided as necessary under each request.

|      |
|------|
| CDFN |
|------|

#### 9.4.1 .CDFN

The .CDFN request is used to redefine the number of I/O channels. Each job, whether foreground or background, is initially provided with 16(decimal) I/O channels. .CDFN allows the number to be expanded to as many as 256 (decimal) channels.

Note that .CDFN defines new channels; the previously-defined channels are not used. Thus, a .CDFN for 20(decimal) channels (while the 16



## Programmed Requests

original channels are defined) causes only 20 I/O channels; the space for the original 16 is unused.

Macro Call: .CDFN .area, .addr, .num

where: .addr is the address where the I/O channels begin  
.num is the number of I/O channels to be created

Request Format:

R0 ⇒ .area: 

|       |   |
|-------|---|
| 15    | 0 |
| .addr |   |
| .num  |   |

The space used to contain the new channels is taken from within the user program. Each I/O channel requires 5 words of memory. Thus, the user must allocate 5\*N words of memory, where N is the number of channels to be defined.

It is recommended that the .CDFN request be used at the beginning of a program, before any I/O operations have been initiated. If more than one .CDFN request is used, the channel areas must either start at the same location or not overlap at all. The two requests .SRESET and .HRESET cause the user's channels to revert to the original 16 channels defined at program initiation. Hence, any .CDFNs must be reissued after using those directives.

Errors:

| <u>Code</u> | <u>Explanation</u>   |
|-------------|--|
| 0           | An attempt was made to define fewer channels than already exist. |

Example:

```
      .MCALL  ..V2...REGDEF
      ..V2..
      .REGDEF
START: .MCALL  .CDFN,.PRINT,.EXIT
      .CDFN   #R0LIST,#CHANL,#40.
      BCS     BADCDF
      .PRINT  #MSG1
      .EXIT
BADCDF: .PRINT  #MSG2
      .EXIT
MSG1:   .ASCIZ  /CDFN O.K./
      .EVEN
MSG2:   .ASCIZ  /BAD CDFN/
      .EVEN
R0LIST: .BLKW   3                      /EMPTY ARGUMENT LIST
CHANL:  .BLKW   40.*5                 /ROOM FOR CHANNELS
      .END    START
```

The example defines 40 (decimal) channels to start at location CHANL. An error occurs if 40 or more channels are already defined.



## Programmed Requests

### .CHAIN

#### 9.4.2 .CHAIN

This request allows a background program to pass control directly to another background program without operator intervention. Since this process may be repeated, a large "chain" of programs can be strung together.

The area from locations 500-507 contains the device name and file name (in RAD50) to be chained to, and the area from locations 510-777 is used to pass information between the chained programs.

Macro Call: .CHAIN

#### Notes:

1. No assumptions should be made concerning which areas of memory will remain intact across a .CHAIN. In general, 500-777 is the only area guaranteed to be preserved across a .CHAIN.
2. I/O channels are left open across a .CHAIN for use by the new program. However, I/O channels opened via a .CDFN request are not available in this way. Since the monitor reverts to to the original 16 channels during a .CHAIN, programs which leave files open across a .CHAIN should not use .CDFN. Furthermore, non-resident device handlers are released during a .CHAIN, and must be .FETChed again by the new program.
3. If a program normally loads into the area 500-777, bit 8 of the JSW should be set during program assembly. This causes the monitor to load the area properly. If the bit is not set, locations 500-777 are preserved from the chaining program, causing the new program to malfunction.

#### Errors:

.CHAIN is implemented by simulating the monitor RUN command (described in Chapter 2), and can produce any errors which RUN can produce. If an error occurs, the .CHAIN is abandoned and the Keyboard Monitor is entered.

When using .CHAIN, care should be taken for initial stack placement, since the program being "chained to" is started. The Linker normally defaults the initial stack to 1000(octal); if caution is not observed, the stack may destroy chain data before it can be used.



## Programmed Requests

### Example:

```
      .MCALL  ,,V2,,,REGDEF
      ..V2..
      .REGDEF
      .MCALL  .CHAIN,.TTYIN
START:
      MOV     #500,R1           ;SET UP TO CHAIN
      MOV     #CHPTR,R2        ;DEVICE, FILE NAME TO 500-511
      .REPT   4
      MOV     (R2)+,(R1)+
      .ENDR
LOOP:  .TTYIN                  ;NOW GET A COMMAND LINE
      MOVB    R0,(R1)+         ;AND PASS IT TO THE JOB
      CMPB    R0,#12           ;IN LOCATIONS 512 AND UP
      BNE     LOOP            ;LOOP UNTIL LINE FEED
      CLRB    (R1)+           ;PUT IN A NULL BYTE
      .CHAIN
CHPTR: .RAD50  /DK /
      .RAD50  /TECO /
      .RAD50  /SAV/
      .END     START
```

## .CHCOPY

### 9.4.3 .CHCOPY (F/B only)

The .CHCOPY request opens a channel for input, logically connecting it to a file which is currently open by the other job for either input or output. This request may be used by either the foreground or the background. .CHCOPY must be done before each .READ and .WRITE.

Macro Call: .CHCOPY .area, .chan, .ochan

where: .chan is the channel which the job will use to read the data.

.ochan is the channel number of the other job which is to be copied

### Request Format:

R0 ⇒ .area: 

|    |        |
|----|--------|
| 13 | .chan  |
|    | .ochan |

.CHCOPY is legal only on files which are on disk or DECTape; however, no errors are detected by the system if another device is used.

### Notes:

1. If the other job's channel was opened via an .ENTER in order to create a file, the copier's channel indicates a file which extends to the highest block that the creator of the file had written at the time the .CHCOPY was executed.



## Programmed Requests

2. A channel which is open on a non-file structured device should not be copied, because intermixture of buffer requests may result.
3. A program can write to a file (which is being created by the other job) on a copied channel just as it could if it were the creator. When the copier's channel is closed, however, no directory update takes place.

### Errors:

| <u>Code</u> | <u>Explanation</u>   |
|-------------|--|
| 0           | Other job does not exist, does not have enough channels defined, or does not have the specified channel (.ochan) open. |
| 1           | Channel (.chan) already open.  |

### Example:

In this example, .CHCOPY is used to read data currently being written by the other job. The correct block number and channel to read is obtained by a .RCVDW command. The channel number will be in MSG+4.

```

.MCALL ..V2...REGDEF
..V2..
.REGDEF
.MCALL .CHCOPY,.RCVDW,.PURGE,.READW,.EXIT,.PRINT
ST:
.PURGE #0 ;MAKE SURE WE HAVE CLEAR
;CHANNEL
.RCVDW #AREA,#MSG,#2 ;READ TWO WORDS, BLOCK #
;AND CHANNEL
BCS NOJOB ;NO JOB THERE
.CHCOPY #AREA,#0,MSG+4 ;CHANNEL # IS IN THERE
BCS BUSY ;BUT BUSY
.READW #AREA,#0,#BUFF,#256,,MSG+2 ;GET THE CORRECT BLOCK
BCS RDERR
.PRINT #OKMSG
.EXIT
NOJOB: .PRINT #MSG1
.EXIT
BUSY: .PRINT #MSG2
.EXIT
RDERR: .PRINT #MSG3
.EXIT
AREA: .BLKW 5
MSG1: .BLKW 5
BUFF: .BLKW 256,
MSG1: .ASCIZ /NO JOB!/
MSG2: .ASCIZ /BUSY!/
MSG3: .ASCIZ /READ ERROR/
OKMSG: .ASCIZ /READ OK/
.EVEN

.EXIT
.END ST

```



## Programmed Requests

### .CLOSE

#### 9.4.4 .CLOSE

The .CLOSE request terminates activity on the specified channel and frees it for use in another operation. The handler for the associated device must be in memory.

Macro Call: .CLOSE .chan

Request Format:

R0 ⇒ 7 .chan

A .CLOSE is required on any channel opened for either input or output. A .CLOSE request specifying a channel that is not opened is ignored.

A .CLOSE performed on a file which was opened via .ENTER causes the device directory to be updated to make that file permanent. A file opened via .LOOKUP does not require any directory operations. If the device associated with the specified channel already contains a file with the same name and extension, the old copy is deleted when the new file is made permanent. When an entered file is .CLOSEd, its permanent length reflects the highest block written since it was entered; for example, if the highest block written is block number 0, the file is given a length of 1; if the file was never written, it is given a length of 0. If this length is less than the size of the area which was allocated at .ENTER time, the unused blocks are reclaimed as an empty area on the device.

Errors:

.CLOSE does not return any errors. If the device handler for the operation is not in memory, a fatal monitor error is generated.

Example:

An example which illustrates the .CLOSE request follows the discussion of the .WRITW request in Section 9.4.47.



## Programmed Requests

.CMKT

### 9.4.5 .CMKT (F/B only)

The .CMKT request causes one or more outstanding mark time requests to be cancelled (mark time requests are discussed in Section 9.4.22).

Macro Call: .CMKT .area, .id, .time

where: .id is a number used to identify the request to be cancelled. If .id is not equal to 0, the mark time request with that identifying number is cancelled. If more than one mark time request has the same .id, that with the earliest expiration time is cancelled. If .id = 0, all mark time requests for the issuing job are cancelled.

.time is the pointer to a two-word area in which the Monitor will return the amount of time remaining in the cancelled request. The first word contains the high-order time, the second contains the low-order. If an address of 0 is specified, no value is returned. If .id = 0, the .time parameter is ignored and need not be indicated.

Request Format:

R0 ⇒ .area: 

|       |   |
|-------|---|
| 23    | 0 |
| .id   |   |
| .time |   |

Notes:

1. Cancelling a mark time request frees the associated queue element for other uses.
2. A mark time request can be converted into a timed wait by issuing a .CMKT followed by a .TWAIT, and specifying the same .time area.

Errors:

| <u>Code</u> | <u>Explanation</u>  |
|-------------|---|
| 0           | The .id was not zero; a mark time with that identification number could not be found (implying that the request was never issued or that it has already expired). |



## Programmed Requests

### Example:

See the example following the description of the .MRKT request.

## .CNTXSW

### 9.4.6 .CNTXSW (F/B only)

A context switch is an operation performed when a transition is made from running one job to running the other. The .CNTXSW request is used to specify locations to be included in the context switch.

Macro Call: .CNTXSW .area, .addr

where: .addr is a list of addresses terminated by a zero word. The addresses in the list must be even and:

- a. in the range 2-476, or
- b. in the user job area, or
- c. in the I/O page (addresses 160000-177776).

### Request Format:

R0 ⇒ .area: 

|       |   |
|-------|---|
| 33    | 0 |
| .addr |   |

The system always saves the parameters it needs to uniquely identify and execute a job, including all registers, and the locations:

|       |                             |
|-------|-----------------------------|
| 34/36 | Vector for TRAP instruction |
| 40-52 | System Communication Area   |

If an .SFPA request (Section 9.4.37) has been executed with a non-zero address, all floating point registers and the floating point status are also saved.

It is possible that both jobs may want to share the use of a particular location and that location is not included in normal context switch operations. For example, if a program uses the IOT instruction to perform some internal user function (such as print error messages), it must set up the vector at 20 and 22 to point to an internal IOT trap handling routine. If both foreground and background wish to use IOT, the IOT vector must always point to the proper location for the job which is executing. Including locations 20 and 22 in the .CNTXSW list for both jobs will accomplish this.

If .CNTXSW is issued more than once, only the latest list is used; the previous address list is discarded. Thus, all addresses to be switched must be included in one list. If the address (.addr) is zero, no extra locations are switched. The list may not be in an area



## Programmed Requests

into which the USR swaps, nor may it be modified while a job is running.

### Errors:

| <u>Code</u> | <u>Explanation</u>                                |
|-------------|---|
| 0           | One or more of the above conditions was violated. |

### Example:

In this example, .CNTXSW request is used to specify that locations 20 and 22 (IOT vector) and certain necessary EAE registers be context switched. This allows both jobs to use IOT and the EAE simultaneously yet independently.

```

      .MCALL  ..V2...REGDEF,.CNTXSW,.PRINT,.EXIT
      ..V2...      ;CALL FOR V2 MACROS
      .REGDEF      ;DEFINE REGISTERS
START: MOV      #LIST,R0      ;SET R0 TO OUR OWN LIST
      .CNTXSW  ,#SWAPLS      ;THE LIST OF ADDRS IS
                               ;AT SWAPLS.
      BCC      15
      .PRINT  #ADDERR      ;ADDRESS ERROR(SHOULD NOT
                               ;OCCUR)
      .EXIT
15:   .PRINT  #CNTOK
      .EXIT
SWAPLS: .WORD  20      ;ADDRESSES TO INCLUDE IN LIST
      .WORD  22
      .WORD  177302
      .WORD  177304
      .WORD  177310
      .WORD  0
LIST:  .BYTE  0,33      ;FUNCTION CODE WORD
      .WORD  0      ;THE MACRO FILLS THIS ONE.
ADDERR: .ASCIZ /ADDRESSING ERROR/
      .EVEN
CNTOK:  .ASCIZ /CONTEXT SWITCH O.K./
      .EVEN
      .END  START

```

.CSIGEN

### 9.4.7 .CSIGEN

The .CSIGEN request calls the Command String Interpreter (CSI) in general mode to process a standard RT-11 command string (see Chapter 2 for the description of a standard command string). In general mode, all file .LOOKUPS and .ENTERS as well as handler .FETCHs are



## Programmed Requests

performed. When called in general mode, the CSI closes channels 0-10 (octal).

Macro Call: .CSIGEN .devspc, .defext, .cstring

where: .devspc is the address of the memory area where the device handlers (if any) are to be loaded.

.defext is the address of a four-word block which contains the RAD50 default extensions. These extensions are used when a file is specified without an extension.

.cstring is the address of the ASCIIZ input string or a #0 if input is to come from the console terminal. (In a F/B environment only, if the input is from the console terminal, an .UNLOCK of the USR is automatically performed, even if the USR is locked at the time.) If the string is in memory, it must not contain a <CR><LF>, but must terminate with a zero byte. If the .cstring field is left blank, input is automatically taken from the console terminal.

.CSIGEN loads all necessary handlers and opens the files as specified. The area specified for the device handlers must be large enough to hold all the necessary handlers simultaneously. If the device handlers exceed the area available, the user program may be destroyed. The system, however, is protected from this.

When the FMT is complete, register 0 points to the first available location above the handlers.

The four-word block pointed to by .defext is arranged as:

Word 1: default extension for all input channels

Words 2,3,and 4: default extensions for output channels 0,1,2 respectively

If there is no default for a particular position, the associated word must contain a zero. All extensions are expressed in Radix 50. For example, the following block can be used to set up default extensions for a macro assembler:

```
DEFEXT: .RAD50 "MAC"
        .RAD50 "OBJ"
        .RAD50 "LST"
        .WORD 0
```

In the command string:

\*DT0:ALPHA,DT1:BETA=DT2:INPUT

the default extension for input is MAC; for output, OBJ and LST. If no = sign is present, input is assumed.

When control returns to the user program after a call to .CSIGEN, all the specified files have been opened for input and/or output. The association is as follows: the three possible output files are



## Programmed Requests

assigned to channels 0, 1, and 2; the six input slots are assigned to channels 3 through 10. A null specification causes the associated channel to remain inactive. For example, in the following string:

`*,LP:=F1,F2`

channel 0 is inactive since the first slot is null. Channel 1 is associated with the line printer, and channel 2 is inactive. Channels 3 and 4 are associated with two files on DK:, while channels 5 through 10 are inactive. The user program can determine whether a channel is inactive by issuing a .WAIT request on the associated channel, which returns an error if the channel is not open.

Switches and their associated values are returned on the stack; see Section 9.4.8.1 for a description of the way switch information is passed.

### Errors:

If CSI errors occur and input was from the console terminal, an error message describing the fault is printed on the terminal and the CSI retries the command (these messages appear in Section 9.4.8.1). If the input was from a string, the carry bit is set and byte 52 contains the error code. The errors are:

| <u>Code</u> | <u>Explanation</u>  |
|-------------|---|
| 0           | Illegal command (bad separators, illegal filename, command too long, etc.). |
| 1           | A device specified is not found in the system tables.                       |
| 2           | Unused.   |
| 3           | An attempt to .ENTER a file failed because of a full directory.             |
| 4           | An input file was not found in a .LOOKUP.                                   |

### Example:

This example uses the general mode of the CSI in a program to copy an input file to an output file. Command input to the CSI is from the console terminal.

```
      ,MCALL  ..V2... ,REGDEF
      ..V2..
      ,REGDEF
      ,MCALL  .CSIGEN, .READW, .PRINT, .EXIT, .WRITW, .CLOSE, .SRESET
ERRWD=52

START: .CSIGEN #DSPACE, #DEXT      ;GET STRING FROM TERMINAL
      MOV     R0, BUFF            ;R0 HAS FIRST FREE LOCATION
      CLR     INBLK               ;INPUT BLOCK #
      MOV     #LIST, R5           ;EMT ARGUMENT LIST
READ:  .READW  R5, #3, BUFF, #256, INBLK ;READ CHANNEL 3
      BCC     2$                  ;NO ERRORS
      TSTB    #ERRWD              ;EOF ERROR?
      BEQ     EOF                 ;YES
      MOV     #INERR, R0
```



## Programmed Requests

```

131      .PRINT                                ;ERROR MESSAGE
        CLR      R0                            ;HARD EXIT
        .EXIT
231      .WRTW   R5,#0,BUFF,#256,,INBLK ;WRITE THE BLOCK
        BCC      NOERR                        ;NO ERROR WRITING
        MOV      #WTERR,R0
        BR       18                            ;HARD OUTPUT ERROR
NOERR:    INC     INBLK                        ;GET NEXT BLOCK
        BR       READ                          ;LOOP UNTIL DONE
EOF:      .CLOSE #0                            ;CLOSE OUTPUT CHANNEL
        .CLOSE #3                            ;AND INPUT CHANNEL
        .SRESET                                ;RELEASE HANDLER FROM MEMORY
        BR       START                        ;GO FOR NEXT COMMAND LINE
DEXT:     .WORD   0,0,0,0                      ;NO DEFAULT EXTENSIONS
BUFF:     .WORD   0                            ;I/O BUFFER START
INBLK:    .WORD   0                            ;RELATIVE BLOCK TO READ/WRITE
LIST:     .BLKW   5                            ;EMT ARGUMENT LIST
INERR:    .ASCIZ  /INPUT ERROR/
        .EVEN
WTERR:    .ASCIZ  /OUTPUT ERROR/
        .EVEN
DSPACE:    .END                                ;HANDLER SPACE
        START

```

In the Command String Interpreter's general mode, the .CLOSEs on channels 0 and 3 are not absolutely necessary since the CSI .CLOSEs channels 0-8 when it is called. However, if .CSISPC is used, or if the line is not obtained using the CSI at all, the .CLOSE calls are required.

### .CSISPC

#### 9.4.8 .CSISPC

The .CSISPC request calls the Command String Interpreter in special mode to parse the command string and return file descriptors and switches to the program. In this mode, the CSI does not perform any handler fetches, .CLOSEs, .ENTERs, or .LOOKUPs.

Macro Call: .CSISPC .outspc, .defext, .cstring

where: .outspc is the address of the 39-word block to contain the file descriptors produced by .CSISPC. This area may overlay the space allocated to .cstring if desired.

.defext is the address of a four-word block which contains the RAD50 default extensions. These extensions are used when a file is specified without an extension.



## Programmed Requests

`.cstring` is the address of the ASCIIZ input string or a #0 if input is to come from the console terminal. If the string is in memory, it must not contain a <CR><LF> but must terminate with a zero byte. If `.cstring` is blank, input is automatically taken from the console terminal.

The 39-word file description consists of nine file descriptor blocks (five words for each of three possible output files; four words for each of six possible input files) which correspond to the nine possible files (three output, six input). If any of the nine possible filenames are not specified, the corresponding descriptor block is filled with zeroes.

The five-word blocks hold four words of RAD50 representing dev:file.ext, and 1 word representing the size specification given in the string. (A size specification is a decimal number enclosed in square brackets [], following the output file descriptor.) For example,

`*DT3:LIST.MAC[15]=PR:`

Using special mode, the CSI returns in the first five word slot:

|       |                             |
|-------|-----------------------------|
| 16101 | .RAD50 for DT3              |
| 46173 | .RAD50 for LIS              |
| 76400 | .RAD50 for T                |
| 50553 | .RAD50 for MAC              |
| 00017 | Octal value of size request |

In the fourth slot (starting at an offset of 36 (octal) bytes into `.outspc`), the CSI returns:

|       |               |
|-------|---------------|
| 63320 | .RAD50 for PR |
| 0     | No file name  |
| 0     | Specified     |
| 0     |               |

Since this is an input file, only four words are returned.

Switches and their associated values are returned on the stack. See Section 9.4.8.1.

### Errors:

Errors are the same as in general mode. However, since `.LOOKUPS` and `.ENTERS` are not done, the error codes which are valid are:

| <u>Code</u> | <u>Explanation</u>   |
|-------------|----------------------|
| 0           | Illegal command line |
| 1           | Illegal device       |

### Example:

This example illustrates the use of the special mode of CSI. This example could be a program to read a file which is not in RT-11 format to a file under RT-11.



## Programmed Requests

```

        ,MCALL  ..V2,..,REGDEF
        ..V2..
        ,REGDEF
        ,MCALL  ,CSI3PC,,PRINT,,EXIT,,ENTER,,CLOSE

START:  ,CSI3PC  #OUTSPC,#DEXT,#CSTRNG ;GET INPUT FROM A
                                           ;STRING IN MEMORY

        BCC     28
        MOV     #SYNERR,R0             ;SYNTAX ERROR
13:     ,PRINT                                     ;ERROR MESSAGE
        ,EXIT
23:     ,ENTER  #LIST,#0,#OUTSPC,#64. ;ENTER FILE UNDER RT-11
        BCC     38
        MOV     #ENMSG,R0             ;ENTER FAILED
        BR      18
33:     JBR     R5,INPUT                ;ROUTINE INPUT WILL USE
                                           ;THE INFORMATION AT
                                           ;#OUTSPC+36 TO READ INPUT
                                           ;FROM THE NON-RT11 DEVICE.
                                           ;INPUT IS PROCESSED AND
                                           ;WRITTEN VIA ,WRITW REQUESTS
                                           ;MAKE OUTPUT FILE PERMANENT
                                           ;AND EXIT PROGRAM
        ,CLOSE  #0
        ,EXIT
CSTRNG: ,ASCIZ  "DT4:IRTFIL.MAC=DT2:IDOS.MAC"
        ,EVEN
DEXT:   ,WORD   0,0,0,0                ;NO DEFAULT EXTENSIONS
LIST:   ,BLKW   5                      ;LIST FOR EMT CALLS
SYNERR: ,ASCIZ  "CSI ERROR"
ENMSG:  ,ASCIZ  "ENTER FAILED"
        ,EVEN
INPUT:  RTS     R5
OUTSPC: ,
        ,END     START

```

### 9.4.8.1 Passing Switch Information

In both general and special modes of the CSI, switches and their associated values are returned on the stack. A CSI switch is a slash (/) followed by any character. The CSI does not restrict the switch to printing characters, although it is suggested that printing characters be used wherever possible. The switch can be followed by an optional value, which is indicated by a : or ! separator. The : separator is followed by either an octal number or by one to three alphanumeric characters, the first of which must be alphabetic, which are converted to Radix-50. The ! separator is followed by a decimal value. Switches can be associated with files with the CSI. For example:

```
*DK:FOO/A,DT4:FILE.OBJ/A:100
```

In this case, there are two A switches. The first is associated with the input file DK:FOO. The second is associated with the input file DT4:FILE.OBJ, and has a value of 100(8). The stack output of the CSI is as follows:



## Programmed Requests

| <u>Word #</u>       | <u>Value</u>                 | <u>Meaning</u>  |
|---------------------|------------------------------|---|
| 1<br>(top of stack) | N                            | Number of switches found in command string. If N=0, no switches were found.   |
| 2                   | Switch value and file number | Even byte = 7-bit ASCII switch value.<br>Bits 8-14 = Number (0-10) of the file with which the switch is associated.<br>Bit 15 = 1 if the switch had a value.<br>= 0 if the switch had no value. |
| 3                   | Switch value or next switch  | If word 2 was less than 0, word 3 = switch value. If word 2 was greater than 0, this word is the next switch value (if it exists).  |

For example, if the input to the CSI is:

\*FILE/B:20,FIL2/E=DT3:INPUT/X:SY:20

on return, the stack is:

|                |        |   |
|----------------|--------|---|
| Stack Pointer→ | 3      | Three switches appeared.                          |
|                | 101530 | Last switch=X; with file 3, has a value.          |
|                | 20     | Value of switch X=20                              |
|                | 101530 | Next switch =X; with file 3, has a value.         |
|                | 075250 | Next value of switch X=RAD50 code for SY.         |
|                | 505    | Next switch=E; associated with file 1, no value.  |
|                | 100102 | Switch=B; associated with file 0 and has a value. |
|                | 20     | Value is 20.                                      |

As an extended example, assume the following string was input for the CSI in general mode:

\*FILE[8],LP:,SY:FILE2[20]=PR:,DT1:IN1/B,DT2:IN2/M:7

Assume also that the default extension block is:

|         |        |       |                          |
|---------|--------|-------|--------------------------|
| DEFEXT: | .RAD50 | 'MAC' | ;INPUT EXTENSION         |
|         | .RAD50 | 'OP1' | ;FIRST OUTPUT EXTENSION  |
|         | .RAD50 | 'OP2' | ;SECOND OUTPUT EXTENSION |
|         | .RAD50 | 'OP3' | ;THIRD OUTPUT EXTENSION  |

The result of this CSI call would be:

1. A file named FILE.OP1 is entered on channel 0 on device DK; channel 1 is open for output to the device LP; a 20-block file named FILE2.OP3 is entered on the system device on channel 2.



## Programmed Requests

2. Channel 3 is open for input from paper tape; channel 4 is open for input from a file IN1.MAC on device DT1; channel 5 is open for input from IN2.MAC on device DT2.

3. The stack contains switches and values as follows:

|        |
|--------|
| 2      |
| 102515 |
| 7      |
| 2102   |

### Explanation

2 switches found in string.  
Second switch is M, associated with Channel 5; has a numeric value.  
Numeric value is 7.  
Switch is B, associated with Channel 4; has no numeric value.

If the CSI were called in special mode (Section 9.4.8), the stack would be the same as for the general mode call, and the descriptor table would contain:

|          |       |                         |               |
|----------|-------|-------------------------|---------------|
| .OUTSPC: | 15270 | ;.RAD50                 | 'DK'          |
|          | 23364 | ;.RAD50                 | 'FIL'         |
|          | 17500 | ;.RAD50                 | 'E'           |
|          | 60137 | ;.RAD50                 | 'OP1'         |
|          | 10    | ;.LENGTH OF             | 8 BLOCKS      |
|          | 46600 | ;.RAD50                 | 'LP'          |
|          | 0     | ;.NO NAME OR LENGTH     | SPECIFIED     |
|          | 0     |                         |               |
|          | 0     |                         |               |
|          | 0     |                         |               |
|          | 75250 | ;.RAD50                 | 'SY'          |
|          | 23364 | ;.RAD50                 | 'FIL'         |
|          | 22100 | ;.RAD50                 | 'E2'          |
|          | 60141 | ;.RAD50                 | 'OP3'         |
|          | 24    | ;.LENGTH OF             | 20 (DECIMAL)  |
|          | 63320 | ;.RAD50                 | 'PR'          |
|          | 0     |                         |               |
|          | 0     |                         |               |
|          | 0     |                         |               |
|          | 16077 | ;.RAD50                 | 'DT1'         |
|          | 35217 | ;.RAD50                 | 'IN1'         |
|          | 0     | ;.RAD50                 | ' '           |
|          | 50553 | ;.RAD50                 | 'MAC'         |
|          | 16100 | ;.RAD50                 | 'DT2'         |
|          | 35220 | ;.RAD50                 | 'IN2'         |
|          | 0     | ;.RAD50                 | ' '           |
|          | 50553 | ;.RAD50                 | 'MAC'         |
|          | 0     |                         |               |
|          | .     |                         |               |
|          | .     |                         |               |
|          | .     |                         |               |
|          | 0     | (twelve more zero words | are returned) |

Keyboard error messages which may occur from incorrect use of the CSI when input is from the console keyboard include:

| <u>Message</u> | <u>Meaning</u>            |
|----------------|---------------------------|
| ?ILL CMD?      | Syntax error.             |
| ?FIL NOT FND?  | Input file was not found. |



## Programmed Requests

?DEV FUL?  
?ILL DEV?

Output file will not fit.  
Device specified does not exist.

### Notes:

1. In many cases, the user program does not need to process switches in CSI calls. However, the user at the console may inadvertently enter switches. In this case, it is wise for the program to save the value of the stack pointer before the call to the CSI, and restore it after the call. In this way, no extraneous values will be left on the stack.
2. In the F/B System, calls to the CSI which require console terminal input will always do an implicit .UNLOCK of the USR. This should be kept in mind when using .LOCK calls.

|        |
|--------|
| .CSTAT |
|--------|

### 9.4.9 .CSTAT (F/B only)

This request furnishes the user with information about a channel. It is supported only in the F/B environment; no information is returned in the Single-Job Monitor.

Macro Call: .CSTAT .area, .chan, .addr

where: .addr is the address of a 6-word block which is to contain the status

### Request Format:

R0 ⇒ .area: 

|       |       |
|-------|-------|
| 27    | .chan |
| .addr |       |

The 6 words passed back to the user are:

1. Channel status word (see Section 9.4.34)
2. Starting block number of file
3. Length of file
4. Highest block written since file was opened
5. Unit number of device with which this channel is associated
6. RAD50 of the device name with which the channel is associated (this is a physical device name, unaffected by any user name ASSIGNment in effect)

The fourth word (highest block) is maintained by the .WRITE requests. If data is being written on this channel, the highest relative block number is kept in this word.



## Programmed Requests

### Errors:

| <u>Code</u> | <u>Explanation</u>       |
|-------------|--------------------------|
| 0           | The channel is not open. |

### Example:

In this example, .CSTAT is used to determine the .RAD50 representation of the device with which the channel is associated.

```

.MCALL ..V2... ,REGDEF, .CSIGEN, .CSTAT
..V2..
.REGDEF

.MCALL .PRINT, .EXIT

STI      .CSIGEN #DEVSDC, #DEFEXT /OPEN FILES
.CSTAT   #AREA, #0, #ADDR  /GET THE STATUS
BCS      NOCHAN           /CHANNEL 0 NOT OPEN
MOV      #ADDR+10, R5      /POINT TO UNIT #
MOV      (R5)+, R0         /UNIT # TO R0
ADD      (PC)+, R0         /MAKE IT RAD50
.RAD50   / 0/
ADD      (R5), R0          /GET DEVICE NAME
MOV      R0, DEVNAM        /DEVNAM HAS RAD50 DEVICE NAME
.EXIT

AREA1    .BLKW 5           /EMT ARG LIST
ADDR1    .BLKW 6           /AREA FOR CHANNEL STATUS
DEVNAM1   .WORD 0          /STORAGE FOR DEVICE NAME
DEFEXT1   .WORD 0,0,0,0
NOCHAN1   .PRINT #MSG
.EXIT

MSG1      .ASCIZ /NO OUTPUT FILE/
.EVEN

DEVSDC=
.END      ST

```

.DELETE

#### 9.4.10 .DELETE

The .DELETE request deletes a named file from an indicated device.

Macro Call: .DELETE .area, .chan, .dblkw, .count



## Programmed Requests

where: .count is used by magtape/cassette only, (Refer to Appendix H for more information concerning the magtape and cassette handlers.)

### Request Format:

R0 → .area: 

|        |       |
|--------|-------|
| 0      | .chan |
| .dblk  |       |
| .count |       |

### Note:

The channel specified in the .DELETE request must not be in use when the request is made, or an error will occur. The file is deleted from the device, and an empty (UNUSED) entry of the same size is put in its place. A .DELETE issued to a non-file structured device is ignored. .DELETE requires that the handler to be used be in memory at the time the request is made. When the .DELETE is complete, the specified channel is left inactive.

### Errors:

| <u>Code</u> | <u>Explanation</u>                         |
|-------------|--|
| 0           | Channel is active                          |
| 1           | File was not found in the device directory |

### Example:

This example uses the special mode of CSI to delete files.

```

.MCALL ..V2...REGDEF
..V2..
.REGDEF
.MCALL .SRESET,.CSISPC,.DELETE,.PRINT,.EXIT

START: .SRESET                                ;MAKE SURE CHANNELS
                                           ;ARE FREE
        .CSISPC #OUTSPC,#DEFEXT             ;GET COMMAND LINE
                                           ;TERMINAL DIALOG WAS
                                           ;DTIFILE
        .DELETE #LIST,#0,#INSPC             ;USE CHANNEL 0 TO
                                           ;DELETE THE FILE
                                           ;WHICH IS AT THE
                                           ;FIRST INPUT SLOT.
                                           ;OK, LOOP AGAIN
        BCC      START                     ;NO SUCH FILE
        .PRINT  #NOFILE
        BR       START
NOFILE: .ASCIZ  /FILE NOT FOUND/
        .EVEN
DEFEXT: .RAD50  /MAC/                      ;.MAC INPUT EXTENSION
        .WORD   0,0,0                     ;NO OUTPUT DEFAULTS
LIST:   .BLKW   2                          ;EMT ARG LIST
OUTSPC=.
INSPC=+.36
        .BLKW   39.
        .END      START

```

INSPC is the address of the first input slot in the CSI input table.



## Programmed Requests

### **.DEVICE**

#### 9.4.11 .DEVICE (F/B Only)

This request allows the user to set up a list of addresses to be loaded with specified values when a program is terminated. Upon an .EXIT or CTRL C, this list is picked up by the system and the appropriate addresses are set up with the corresponding values. This function is primarily designed to allow user programs to load device registers with necessary values. In particular, it is used to turn off a device's interrupt enable bit when the program servicing the device terminates.

Macro Call: .DEVICE .area, .addr

where: .addr is the address of the list of masks and words.

Request Format:

R0 ⇒ .area: 

|       |   |
|-------|---|
| 14    | 0 |
| .addr |   |

The list is composed of address/value pairs and should be terminated by a 0 address. Only one list can be active at a given time. If multiple .DEVICE requests are given, the last list specified is the one used.

Note:

When the job is terminated for any reason, the list is scanned once. At that point, the monitor disables the feature until another .DEVICE call is executed. Thus, background programs which are re-enterable should include .DEVICE as a part of the reenter code.

Errors:

None.

Example:

The following example shows how .DEVICE is used to disable interrupts from the AFC11 (A-D converter sub-system).

```
                .MCALL  ..V2,,,REGDEF
                ..V2,,
                .REGDEF
                .MCALL  .DEVICE,.EXIT

START:  .DEVICE #LIST
        .EXIT
LIST:   .BYTE  0,14                JEMT ARG LIST
```



## Programmed Requests

```
ATOD:      .WORD      ATOD
           172570
           0
           0
           .END      START

           IADDRESS IS 172570
           IJAM A 0 INTO IT
           ITHIS 0 TERMINATES THE LIST.
```

### .DSTATUS

#### 9.4.12 .DSTATUS

This request is used to obtain information about a particular device.

Macro Call: .DSTATUS .cblk, .devnam

where: .cblk is the 4-word space used to store the status information.

.devnam is the pointer to the RAD50 device name.

.DSTATUS looks for the device specified by .devnam and, if found, returns four words of status starting at the address specified by .cblk. The four words returned are:

##### 1. Status Word

Bits 7-0: contain a number which identifies the device in question. The values (octal) currently defined are:

- 0 = RK05 Disk
- 1 = TC11 DECTape
- 2 = Unused
- 3 = Line Printer
- 4 = Console Terminal (LT33, LT35, LA30, LA36, VT05, VT50)
- 5,6 = Unused
- 7 = PC11 High-speed Reader
- 10 = PC11 High-speed Punch
- 11 = Magtape
- 12 = RF11 Disk
- 13 = TA11 Cassette
- 14 = Card Reader

- Bit 15: 1= File structured device (disk, DECTape)  
0= Non-file device
- Bit 14: 1= Read-only device
- Bit 13: 1= Write-only device
- Bit 12: 1= Device whose directory is not an RT-11 directory (magtape, cassette).



## Programmed Requests

### 2. Handler size.

The size of the device handler, in bytes.

### 3. Entry point.

Non-zero implies the handler is now in memory; zero implies it must be .FETCHed before it can be used.

### 4. Device size.

The size of the device (in 256-word blocks) for file structured devices; zero for non-file devices.

The device name may be a user-assigned name.

## Errors:

| <u>Code</u> | <u>Explanation</u>          |
|-------------|-----------------------------|
| 0           | Device not found in tables. |

## Example:

This example shows how to determine if a particular device handler is in memory and, if it is not, how to .FETCH it there.

```

.MCALL ..V2... ,REGDEF
..V2..
.REGDEF
.MCALL .DSTATUS, .PRINT, .EXIT, .FETCH

START: .DSTATUS #CORE, #FPTR      ;GET STATUS OF DEVICE
      BCC      18                ;
      .PRINT   #ILLDEV           ;DEVICE NOT IN TABLES
      .EXIT
18:    TST      CORE+4            ;IS DEVICE RESIDENT?
      BNE      28                ;
      .FETCH   #HNDLR, #FPTR     ;NO, GET IT
      BCC      28                ;
      .PRINT   #FEFAIL           ;FETCH FAILED
      .EXIT
28:    .PRINT   #FECHOK
      .EXIT
CORE:  .BLKW    4                ;STATUS GOES HERE
FPTR:  .RAD50   /DT0/            ;DEVICE NAME
      .RAD50   /FILE MAC/       ;FILE NAME
FEFAIL: .ASCIZ   /FETCH FAILED/
ILLDEV: .ASCIZ   /ILLEGAL DEVICE/
      .EVEN
FECHOK: .ASCIZ   /FETCH O.K./
      .EVEN
HNDLR: .          ;HANDLER WILL GO HERE
      .END      START

```



## Programmed Requests

.ENTER

### 9.4.13 .ENTER

The .ENTER request allocates space on the specified device and creates a tentative entry for the named file. The channel number specified is associated with the file.

Macro Call: .ENTER .area, .chan, .dblk, .length, .count

where: .length is the file specification. The file allocation is as follows:

0 - either 1/2 the largest empty segment or the entire second largest segment, whichever is largest. (A maximum size for non-specific .ENTERS may be patched in the monitor.)

M - a file of M blocks. M may exceed the maximum mentioned above.

-1 - the largest empty segment on the device.

.count file number for magtape/cassette (see Appendix H); if this argument is blank, a value of zero is assumed.

#### Request Format:

R0 ⇒ .area: 

|   |         |
|---|---------|
| 2 | .chan   |
|   | .dblk   |
|   | .length |
|   | .count  |

The file created with an .ENTER is not a permanent file until the .CLOSE on that channel is given. Thus, the newly created file is not available to .LOOKUP and the channel may not be used by .SAVSTATUS requests. However, it is possible to go back and read data which has just been written into the file by referencing the appropriate block number. When the .CLOSE to the channel is given, any already existing permanent file of the same name on the same device is deleted and the new file becomes permanent. Although space is allocated to a file during the .ENTER operation, the actual length of the file is determined when .CLOSE is requested.

Each job may have up to 256 files open on the system at any time. If required, all 256 may be opened for output with the .ENTER function. .ENTER requires that the device handler be in memory when the request is made. Thus, a .FETCH should normally be executed before a .ENTER



## Programmed Requests

can be done. On return, R0 contains the size of the area actually allocated for use.

### Notes:

When using the 0 length feature of .ENTER, it must be kept in mind that less than the largest empty space is allocated. This can have an important effect in transferring files between devices (particularly DECTape) which have a relatively small capacity. For example, to transfer a 200-block file to a DECTape on which the largest available empty space is 300 blocks, a 0 length transfer will not work. Since the .ENTER allocates half the largest space, only 150 blocks are really allocated and an output error will occur during the transfer. If a specific length of 200 is requested, however, the transfer will proceed without error.

### Errors:

| <u>Code</u> | <u>Explanation</u>  |
|-------------|---|
| 0           | Channel is in use.  |
| 1           | In a fixed length request, no space greater than or equal to N was found, or in a non-specific request the device or the directory was found to be empty. |

### Example:

.ENTER may be used to open a file on a specified device, and then write data from memory into that file as follows:

```

.MCALL ..V2...REGDEF,.ENTER,.WRITW,.CLOSE,.PRINT
.MCALL .SRESET,.EXIT,.FETCH
..V2..
.REGDEF

START: .SRESET          ;MAKE SURE ALL CHANNELS
                        ;ARE CLOSED.
        .FETCH  #CORSPC,#FPRT ;FETCH DEVICE HANDLER
BCS      BADFET      ;.FETCH ERROR, PROBABLY
                        ;ILLEGAL DEVICE.
        .ENTER  #AREA,#0,#FPRT ;OPEN A FILE ON THE DEVICE
                        ;SPECIFIED. LENGTH 0 WILL
                        ;GIVE 1/2 OF LARGEST EMPTY
                        ;SPACE NOW AVAILABLE.
BCS      BADENT      ;FAILED. CHANNEL PROBABLY BUSY
        .WRITW  #AREA,#0,#BUFF,#END-BUFF/2,#0
                        ;WRITE DATA FROM MEMORY. THE
                        ;SIZE IS # OF WORDS BETWEEN
                        ;BUFF AND END. START AT BLOCK 0.
BCS      BADWRT      ;WRITE FAILURE.
        .CLOSE  #0      ;CLOSE THE FILE
        .EXIT                                ;AND GO TO KEYBOARD MONITOR.
FPRT:    .RAD50  /DK /      ;FILE WILL BE ON DK
        .RAD50  /FILE EXT/ ;NAMED FILE,EXT
AREA:    .BLKW   10        ;EMT ARGUMENT LIST
BADFET:  .PRINT  #FMSG

```



## Programmed Requests

```

      .EXIT
BADENT: .PRINT #MSG
      .EXIT
BADWRT: .PRINT #WMSG
      .EXIT
FMSG:  .ASCIZ /BAD FETCH/
EMSG:  .ASCIZ /BAD ENTER/
WMSG:  .ASCIZ /WRITE ERROR/
      .EVEN
CORSPC: .BLKW 400          ;LEAVE 400(8) WORDS
                                ;FOR DEVICE HANDLER.
BUFF:
      .REPT 400          ;THIS IS BUFFER TO BE WRITTEN OUT
      .WORD 0,1
      .ENDR
END:
      .END START

```

.EXIT

### 9.4.14 .EXIT

The .EXIT request causes the user program to terminate. When used from a background job under the F/B Monitor and when used under the Single-Job Monitor, .EXIT causes KMON to run in the background area. All outstanding mark time requests are cancelled. Any I/O requests and completion routines pending for that job are allowed to complete. If part of the background job resides where KMON and USR are to be read, the user job is written onto system device scratch blocks. KMON and USR are then loaded and control goes to KMON in the background area. If R0=0 when the .EXIT is done, an implicit INIT command is executed when KMON is entered, disabling the subsequent use of .REENTER, .START, or .CLOSE.

.EXIT also resets any .CDFN and .QSET calls that were done and executes an .UNLOCK if a .LOCK has been done. Thus, the .CLOSE command from the Keyboard Monitor does not operate for programs which perform .CDFN requests.

In a F/B system, an .EXIT from a completion routine acts as if a double CTRL C has been typed, aborting all I/O in progress before exiting. In general, .EXIT from a completion routine should be avoided.

Macro Call: .EXIT

Errors:

None.



## Programmed Requests

### .FETCH

#### 9.4.15 .FETCH

The .FETCH request loads device handlers into memory from the system device.

Macro Call: .FETCH .coradd, .devname

where: .coradd is the address where the device handler is to be loaded.

.devname is the pointer to the RAD50 device name.

The storage address for the device handler is passed on the stack. When the .FETCH is complete, R0 points to the first available location above the handler. If the handler is already in memory, R0 keeps the same value as was initially pushed onto the stack. If the argument on the stack is less than 400(8), it is assumed that a handler .RELEAS is being done. (.RELEAS does not dismiss a handler which was .LOADED from the KMON; an .UNLOAD must be done.) After a .RELEAS, a .FETCH must be issued in order to use the device again.

Several requests require a device handler to be in memory for successful operation. These include:

|         |        |         |
|---------|--------|---------|
| .CLOSE  | .READC | .READ   |
| .LOOKUP | .WRITC | .WRITE  |
| .ENTER  | .READW | .SPFUN  |
| .RENAME | .WRITW | .DELETE |

Since foreground jobs must have handlers resident, a .FETCH from the foreground will give a fatal error if the handler has not been previously .LOADED.

#### Errors:

| <u>Code</u> | <u>Explanation</u>  |
|-------------|---|
| 0           | The device name specified does not exist, or there is no handler for that device in the system. |

#### Example:

In the following example, the PR and PP handlers are fetched into memory in preparation for their use by a program. The program sets aside handler space from its free memory area.



## Programmed Requests

```

.MCALL ..V2...REGDEF,.FETCH,.PRINT,.EXIT
..V2..
.REGDEF

START:
.FETCH FREE,#PRNAME    /FETCH PR HANDLER
BCS FERR               /FETCH ERROR
MOV R0,R2
.FETCH R2,#PPNAME      /FETCH PP HANDLER
                        /IMMEDIATELY FOLLOWING
                        /PR HANDLER, R0 POINTS
                        /TO THE TOP OF PR
                        /HANDLER ON RETURN
                        /FROM THAT CALL,
BCS FERR               /NO PP HANDLER
MOV R0,FREE            /UPDATE FREE MEMORY
                        /POINTER TO POINT TO
                        /NEW BOTTOM OF FREE
                        /AREA(TOP OF HANDLERS),

.PRINT #OK
.EXIT
OK: .ASCIZ /FETCH O.K./
.EVEN
FERR: .PRINT #MSG       /PRINT ERROR MESSAGE
.EXIT /AND EXIT
HALT
PRNAME: .RAD50 "PR "     /DEVICE NAMES
PPNAME: .RAD50 "PP "
MSG: .ASCIZ "DEVICE NOT FOUND" /ERROR MESSAGE
.EVEN
FREE: .+2              /POINTER TO FREE MEMORY
.END START

```

.GTIM

### 9.4.16 .GTIM

.GTIM allows user programs to access the current time of day. The time is returned in two words, and is given in terms of clock ticks past midnight.

Macro Call: .GTIM .area, .addr

where: .addr is a pointer to the two words of time to be returned.

Request Format:

R0 ⇒ .area: 

|       |   |
|-------|---|
| 21    | 0 |
| .addr |   |



## Programmed Requests

The high-order time is returned in the first word, the low-order time in the second word. User programs must make the conversion from clock ticks to hours-minutes-seconds. The basic clock frequency (50 or 60 Hz) may be determined from the configuration word in the monitor (see Section 9.2.6). The time of day is not reset at 24:00.

Errors:

None.

Example:

```
      .MCALL  ..V2...,REGDEF,,GTIM,,EXIT
      ..V2,,
      .REGDEF

START:  .GTIM  #LIST,#TIME

      .EXIT
TIME:   .WORD  0,0           ;LOW AND HI ORDER TIME
                                ;RETURNED HERE.

LIST:   .BLKW  2             ;ARGUMENTS FOR THE EMT
      .END  START
```

**.GTJB**

### 9.4.17 .GTJB

The .GTJB request passes certain job parameters back to the user program.

Macro Call: .GTJB .area, .addr

where: .addr is the address of an eight-word block into which the parameters are passed. The values returned are:

- Word 1 - Job Number. 0=Background, 2=Foreground
- 2 - High memory limit
- 3 - Low memory limit
- 4 - Beginning of I/O channel space
- 5-8 - Reserved for future use

Request Format:

R0 ⇒ .area: 

|       |   |
|-------|---|
| 20    | 0 |
| .addr |   |



## Programmed Requests

In the Single-Job Monitor, the job number is always 0 and the low limit 0.

In the F/B Monitor, the job number can either be 0 or 2. If the job number equals 0 (background job), word 2 equals 0 and word 4 describes where the I/O channel words begin. This is normally an address within the Resident Monitor. When a .CDFN is executed, however, the start of the I/O channel area changes to the user specified area.

### Errors:

None.

### Example:

Use .GTJB to determine if this program is executing as a foreground or background job.

```
.MCALL ..V2,,,REGDEF,.GTJB,.PRINT,.EXIT
..V2..
.REGDEF

START:
.GTJB  #LIST,#JOBARG  ;R0 POINTS TO 1ST WORD ON
                        ;RETURN FROM CALL.
MOV    #FMSG,R1
TST    JOBARG          ;BACKGROUND?
BNE    1$              ;NO, PRINT FMSG
MOV    #BMSG,R1
1$     .PRINT  R1
      .EXIT

FMSG:  .ASCIZ  /PROGRAM IN FOREGROUND/
BMSG:  .ASCIZ  /PROGRAM IN BACKGROUND/
      .EVEN

LIST:  .BLKW   2          ;ARGUMENTS FOR THE EMT
JOBARG: .BLKW   8.        ;JOB PARAMETERS PASSED BACK HERE.
      .END    START
```

|             |
|-------------|
| .HERR/.SERR |
|-------------|

### 9.4.18 .HERR/.SERR

.HERR and .SERR are complimentary requests used to govern monitor behavior for serious error conditions. During program execution, certain error conditions may arise which cause the executing program to be aborted (for example, trying to pass I/O to a device which has no handler in memory, or trying to load a device handler over the USR). Normally, these errors cause program termination with one of the



## Programmed Requests

?M- error messages. However, in certain cases it is not feasible to abort the program because of these errors; for example, a multi-user program must be able to retain control and merely abort the user who has generated the error. .SERR accomplishes this by inhibiting the monitor from aborting the job. Instead, it causes an error return to the offending EMT to be taken. On return from that request, the C bit is set and byte 52 contains a negative value indicating the error condition which occurred.

.HERR turns off user error interception and allows the system to abort the job on fatal errors and generate an error message. (.HERR is the default case.)

Macro Calls: .HERR

.SERR

## Errors:

Following is a list of the errors which are returned if soft error recovery is in effect:

| <u>Code</u> | <u>Explanation</u>   |
|-------------|--|
| -1          | Called USR from completion routine.  |
| -2          | No device handler; this operation needs one.   |
| -3          | Error doing directory I/O.   |
| -4          | FETCH error. Either an I/O error occurred while reading the handler, or tried to load it over USR or RMON. |
| -5          | Error reading an overlay.  |
| -6          | No more room for files in the directory.   |
| -7          | Illegal address (F/B only).  |
| -10         | Illegal channel number; number is greater than actual number of channels which exist.                      |
| -11         | Illegal EMT; an illegal function code has been decoded.  |

Traps to 4 and 10, and floating point exception traps are not inhibited. These errors have their own recovery mechanism. (See Section 9.4.42.)

## Example:

This example causes a normally fatal error to generate errors back to the user program. The error returned is used to print an appropriate message.

```

.MCALL ..V2,,,REGDEF,.FETCH,.ENTER,.HERR,.SERR
.MCALL .EXIT,.PRINT
..V2..
.REGDEF

STI    .SERR                                ;TURN ON SOFTWARE ERROR
                                           ;RETURNS
       .FETCH #HDLR,#PTR                    ;GET A DEVICE HANDLER
BCS    FCHERR
.ENTRY #AREA,#1,#PTR                        ;OPEN A FILE ON CHANNEL 1
BCS    ENERR

```



## Programmed Requests

```

        .HERR                                /NOW PERMIT ?M-ERRORS.
        .EXIT

FCHERR: MOV8    #52,R0                      /WAS IT FATAL
        BMI     FTLERR                      /YES
        .PRINT  #FMSG                       /NO... NO DEVICE BY THAT NAME
        .EXIT

ENERR:  MOV8    #52,R0
        BMI     FTLERR
        .PRINT  #EMSG
        .EXIT

FTLERR: NEG     R0                          /THIS WILL TURN POSITIVE
        DEC     R0                          /ADJUST BY ONE
        ASL     R0                          /MAKE IT AN INDEX
        MOV     TBL(R0),R0                  /PUT MESSAGE ADDRESS INTO R0
        .PRINT  TBL(R0)                    /AND PRINT IT.
        .EXIT

TBL:    M1      /CAN'T OCCUR IN THIS PROGRAM
        M2      /NO DEVICE HANDLER IN MEMORY
        M3      /DIRECTORY I/O ERROR
        M4      /FETCH ERROR
        M5      /IMPOSSIBLE FOR THIS PROGRAM
        M6      /NO ROOM IN DIRECTORY
        M7      /ILLEGAL ADDRESS (F/B)
        M10     /ILLEGAL CHANNEL
        M11     /ILLEGAL EMT

M11     /CAN'T OCCUR IN THIS PROGRAM
M21     .ASCIZ  /NO DEVICE HANDLER/
M31     .ASCIZ  "DIRECTORY I/O ERROR"
M41     .ASCIZ  /ERROR DOING FETCH/
M51     /NOT APPLICABLE TO THIS PROGRAM
M61     .ASCIZ  /NO ROOM IN DIRECTORY/
M71     .ASCIZ  /ADDRESS CHECK ERROR/
M101    .ASCIZ  /ILLEGAL EMT/
M111    .ASCIZ  /ILLEGAL DEVICE/
FMSG1   .ASCIZ  /FETCH FAILED/
EMSG1   .ASCIZ  /ENTER FAILED/
        .EVEN

HDLR:   .BLKW   300                          /LEAVE 300 (OCTAL) FOR HANDLER
PTR1    .RAD50  /DT4/                        /DEVICE AND FILE NAME.
        .RAD50  /EXAMPL/
        .RAD50  /MAC/

AREA1   .BLKW   4                            /EMT AREA
        .END   ST

```

.HRESET

### 9.4.19 .HRESET

This request performs the same function as .SRESET, after stopping all I/O transfers in progress for that job. (.HRESET is not used to clear



## Programmed Requests

a hard-error condition.) Note that in the single-job environment, a hardware RESET instruction is used to terminate I/O, while in a F/B environment, only the I/O associated with the job which issued the .HRESET is affected. All other transfers continue.

Macro call: .HRESET

Errors:

None.

Example:

See the example for .SRESET (Section 9.4.40) for format.

### .LOCK/.UNLOCK

#### 9.4.20 .LOCK/.UNLOCK

.LOCK

The .LOCK request is used to "lock" the USR in memory for a series of operations. If all the conditions which cause swapping are satisfied, the user program is read into scratch blocks and the USR is loaded. Otherwise, the USR which is in memory is used, and no swapping occurs. The USR is not released until an .UNLOCK request is given. (Note that in a F/B System, calling the CSI may also perform an implicit .UNLOCK.) A program which has many USR requests to make can .LOCK the USR in memory, make all the requests, and then .UNLOCK the USR; no time is spent doing unnecessary swapping.

In a F/B environment, a .LOCK inhibits the other job from using the USR. Thus, the USR should be locked only as long as necessary.

Macro Call: .LOCK

Note that the .LOCK request reduces time spent in file handling by eliminating the swapping of the USR in and out of memory. If the USR is currently resident, .LOCK is ignored. After a .LOCK has been executed, an .UNLOCK request must be executed to release the USR from memory. The .LOCK/.UNLOCK requests are complimentary and must be matched. That is, if three .LOCK requests are issued, at least three .UNLOCKS must be done, otherwise the USR will not be released. More .UNLOCKS than .LOCKS may occur without error.

Notes:

1. It is vital that the .LOCK call not come from within the area into which the USR will be swapped. If this should occur, the return from the USR request would not be to the user program, but to the USR itself, since the LOCK function inhibits the user program from being re-read.



## Programmed Requests

2. Once a .LOCK has been performed, it is not advisable for the program to destroy the area the USR is in, even though no further use of the USR is required. This causes unpredictable results when an .UNLOCK is done.

Errors:

None.

Example:

See the example following .UNLOCK.

### .UNLOCK

The .UNLOCK request releases the User Service Routine from memory if it was placed there with a .LOCK request. If the .LOCK required a swap, the .UNLOCK loads the user program back into memory. If the USR does not require swapping, the .UNLOCK acts as a no-op.

Macro Call: .UNLOCK

Notes:

1. It is important that at least as many .UNLOCKS are given as .LOCKS. If more .LOCK requests were done, the USR remains locked in memory. It is not harmful to give more UNLOCKS than are required; those that are extra are ignored.
2. The .LOCK/.UNLOCK pairs should be used only when absolutely necessary when running two jobs in the F/B system. When a job .LOCKS the USR, the other job cannot get at it until it is .UNLOCKed. Thus, the USR should not be .LOCKed unnecessarily, as this may degrade performance in some cases.
3. In a F/B System, calling the CSI with input coming from the console terminal performs an implicit .UNLOCK.

Errors:

None.

Example:

This example shows the usage of .LOCK, .UNLOCK, and their interaction with the system.

```
.MCALL ..V2...REGDEF,.LOCK,.UNLOCK,.LOOKUP
.MCALL .SETTOP,.PRINT,.EXIT
..V2..
.REGDEF
```

START:

SYSPTR=54

```
.SETTOP #SYSPTR      ;TRY FOR ALL OF MEMORY
MOV      RD, TOP     ;RD HAS THE TOP
.LOCK                      ;BRING USR INTO MEMORY
.LOOKUP #LIST, #0, #FILE1 ;LOOKUP A FILE ON CHANNEL 0
```



## Programmed Requests

```

231      BCC      18              ;ON ERROR, PRINT A
      .PRINT    #LMSG           ;MESSAGE AND EXIT
      .EXIT
131      MOV      #LIST,R0
      INC        (R0)           ;DO LOOKUP ON CHANNEL 1
      MOV        #FILE2,2(R0)   ;NEW POINTER
      .LOOKUP    ;ALL ARGS ARE FILLED IN
      BCS        28
      .UNLOCK    ;NOW RELEASE USR
      .EXIT

LIST1:   .BLKW    3              ;SPACE FOR ARGUMENTS
FILE1:   .RAD50   /DK /
      .RAD50   /FILE1 MAC/
FILE2:   .RAD50   /DK /
      .RAD50   /FILE2 MAC/
TOP1:    .WORD    0
LMSG1:   .ASCIZ   /LOOKUP ERROR/
      .EVEN

      .END      START

```

In the above example, .SETTOP tries to obtain as much memory as it can. Most likely this will, in a background job, make the USR non-resident (i.e., unless a SET USR NOSWAP command is done at the keyboard). Thus, if the USR were non-resident, swapping must take place for each .LOOKUP given. Using the .LOCK, the USR is brought into memory and remains there until the .UNLOCK is given.

The second .LOOKUP makes use of the fact that the arguments have already been set up at LIST. Thus, it is possible to increment the channel number, put in a new file pointer and then give a simple .LOOKUP, which does not cause any arguments to be moved into LIST.

### .LOOKUP

#### 9.4.21 .LOOKUP

The .LOOKUP request associates a specified channel with a device and/or file, for the purpose of performing I/O operations. The channel used is then "busy" until one of the following requests is executed:

```

.CLOSE
.SAVESTATUS
.SRESET
.HRESET
.PURGE
.CSIGEN    (if channel is in range 0-10 octal)

```



## Programmed Requests

Macro Call: .LOOKUP .area, .chan, .dblk, .count

where: .count is an argument which can optionally be used for the cassette and magtape handlers. Refer to Appendix H for details of this parameter. If .count is blank, a value of zero is assumed.

### Request Format:

R0 → .area:

|   |        |
|---|--------|
| 1 | .chan  |
|   | .dblk  |
|   | .count |

If the first word of the file name in .dblk is zero and the device is a file-structured device, absolute block 0 of the device is designated as the beginning of the "file". This technique allows I/O to any physical block on the device. If a file name is specified for a device which is not file-structured (i.e. PR:FILE.EXT), the name is ignored.

The handler for the selected device must be in memory for a .LOOKUP. On return from the .LOOKUP, R0 contains the length of the file just looked up. If the length returned is 0, either the input device is not RT-11 file structured, or a non-file structured .LOOKUP was done to the device.

### Errors:

| <u>Code</u> | <u>Explanation</u>                          |
|-------------|---|
| 0           | Channel already open.                       |
| 1           | File indicated was not found on the device. |

### Example:

In the following example, the file "DATA.001" on device DT3 is opened for input on channel 7.

```

.MCALL ..V2...REGDEF,.FETCH,.LOOKUP,.PRINT,.EXIT
..V2..
.REGDEF

START:
ERRWD=52
.FETCH #MSPACE,#DT3N ;GET DEVICE HANDLER
BCS FERR ;DT3 IS NOT AVAILABLE
.LOOKUP #LIST,#7,#DT3N ;LOOKUP THE FILE
;ON CHANNEL 7
BCC LDONE ;FILE WAS FOUND
TSTB #ERRWD ;ERROR, WHAT'S WRONG?
BNE NFD ;FILE NOT FOUND
.PRINT #CAMSG ;PRINT 'CHANNEL ACTIVE'
.EXIT
NFD: .PRINT #NFMMSG ;FILE NOT FOUND
.EXIT
CAMSG: .ASCIIZ /CHANNEL ACTIVE/

```



## Programmed Requests

```

NFM8G: .ASCIZ /FILE NOT FOUND/ ;ERROR MESSAGES
DTMSG: .ASCIZ /DT3 NOT AVAILABLE/
      .EVEN
FERR: .PRINT #DTMSG
      .EXIT

LOONE:                                     ;PROGRAM CAN NOW
                                           ;ISSUE READS AND
                                           ;WRITES TO FILE
                                           ;DATA.001 VIA
                                           ;CHANNEL 7

      .EXIT

LIST: .BLKW 5
DT3N: .RAD50 "DT3"                       ;DEVICE
      .RAD50 "DAT"                       ;FILENAME
      .RAD50 "A "                        ;FILENAME
      .RAD50 "001"                       ;EXTENSION

HSPACE:                                     ;RESERVED SPACE FOR DT
      .*,+400                             ;HANDLER

      .END      START

```

## .MRKT

### 9.4.22 .MRKT

The .MRKT request schedules a completion routine to be entered after a specified time interval (clock ticks past midnight) has elapsed.

Macro Call: .MRKT .area, .time, .crtn, .id

where: .time is the pointer to the two words containing the time interval (high-order first; low-order second).

.id is a number assigned by the user to identify the particular request to the completion routine and to any cancel mark time requests. The number need not be unique (i.e., several .MRKT requests may specify the same .id.) On entry to the completion routine, the .id number is in R0.

Request Format:

R0 ⇒ .area:

|       |   |
|-------|---|
| 22    | 0 |
| .time |   |
| .crtn |   |
| .id   |   |



## Programmed Requests

.MRKT requests require a queue element taken from the same list as the I/O queue elements. The element is in use until either the completion routine is entered or a cancel mark time request is issued. The user should allocate enough queue elements to handle at least as many mark time requests as he expects to have pending simultaneously.

### Errors:

| <u>Code</u> | <u>Explanation</u>              |
|-------------|---------------------------------|
| 0           | No queue element was available. |

### Example:

In this example, a mark time is set up to time out an I/O transfer. If the mark time expires before the transfer is done, a message is printed. If the I/O transfer completes before the mark time, the mark time is cancelled. (Note that the example assumes the I/O channel is already open.)

```

.MCALL ..V2,,,REGDEF,,READ,,WAIT,,MRKT,,CMKT
.MCALL .QSET,,PRINT,,EXIT,,LOOKUP
..V2..
.REGDEF

STI      .LOOKUP #AREA,#0,#FILE  ;OPEN A FILE
BCS      LKERR      ;FILE NOT FOUND
MOV      #AREA,=(SP)      ;ENT LIST TO STACK
.QSET    #QUEUE,#5      ;ALLOCATE 5 MORE ELEMENTS
.MRKT    (SP),#INTRVL,#MRTN,#1 ;SET TIMER GOING
BCS      NOMRKT      ;FAILED.
.READ    #RDLST      ;START I/O TRANSFER
BCS      RDERR
.WAIT    #0           ;AND WAIT A WHILE.
.CMKT    (SP),#1      ;SEE IF MARK TIME IS
                        ;DONE.
BCS      NOTDUN      ;FAILED. THAT MEANS THAT
                        ;THE MARK TIME ALREADY
                        ;EXPIRED.

.EXIT

MRTN:    .CMKT    (SP),#1      ;OK, KILL THE TIMER.
        .PRINT    #FAIL      ;DON'T WORRY ABOUT AN
                                ;ERROR HERE.

LKERR:    RTS      PC
        .PRINT    #LM
        .EXIT

RDERR:    .PRINT    #RDMMSG
        .EXIT

NOTDUN:    .PRINT    #FAIL
        .EXIT

NOMRKT:    .PRINT    #NOQ
        .EXIT

NOQ:      .ASCIZ    /NO QUEUE ELEMENTS AVAILABLE/
FAIL:      .ASCIZ    /MARK TIME COMPLETED BEFORE TRANSFER/
LM:        .ASCIZ    /LOOKUP ERROR/
RDMMSG:    .ASCIZ    /READ ERROR/
        .EVEN

INTRVL:    .WORD    0,13.      ;ALLOW 13 CLOCK
                                ;TICKS FOR TRANSFER.

```



## Programmed Requests

```

QUEUE:  .BLKW    5*7           ;AREA FOR QUEUE ELEMENTS
AREA:    .BLKW    5           ;A FEW WORDS FOR EMT LIST
FILE:    .RAD50  /DK FILE  TST/
RDLST:   .BYTE    0           ;CHANNEL 0
        .BYTE    10          ;A READ
BLOCK:   .WORD    0           ;BLOCK #
        .WORD    BUFF        ;BUFFER
        .WORD    256,        ;1 BLOCK
        .WORD    1
BUFF:    .BLKW    256,
        .END      ST

```

## .MWAIT

### 9.4.23 .MWAIT

This request is similar to the .WAIT request. .MWAIT, however, suspends execution until all messages sent by the other job have been transmitted or received. It provides a means for ensuring that a required message has been processed. It should be used primarily in conjunction with the .RCVD or .SDAT modes of message handling, where no action is taken when a message is completed.

Macro Call: .MWAIT

Errors:

None.

Example:

This program requests a message, does some intermediate processing, and then waits until the message is actually sent.

```

        .MCALL    ..V2,,,REGDEF,.MWAIT,.RCVD,.EXIT,.PRINT
        ..V2..
        .REGDEF

WORDS=255,
START:   .RCVD    #AREA,#RBUF,#WORDS ;GET MESSAGE,

                ;INTERMEDIATE PROCESS

        MOV      #RBUF+2,R5
        .MWAIT
        CMPB     (R5)+,#'A           ;MAKE SURE WE HAVE IT,
        BNE      BADMSG              ;FIRST CHARACTER AN A?
                ;NO, INVALID MESSAGE

        .EXIT
BADMSG:  .PRINT   #MSG
        .EXIT

```



## Programmed Requests

```
MSG:      .ASCIZ  /BAD MESSAGE/
AREA:     .BLKW   10
RBUF:     .BLKW   256,
          .EVEN
          .END    START
```

.PRINT

### 9.4.24 .PRINT

The .PRINT request causes output to be printed at the console terminal. When a change occurs in the job producing output, a B> or F> appears. Any text following the message has been printed by the job indicated (foreground or background) until another B> or F> is printed. The string to be printed may be terminated with either a null (0) byte or a 200 byte. If the null (ASCIZ) format is used, the output is automatically followed by a <CR><LF>. If a 200 byte terminates the string, no <CR><LF> is generated.

Macro Call: .PRINT .addr

where: .addr is the address of the string to be printed.

Control returns to the user program after all characters have been placed in the output buffer.

The foreground job issues a message immediately using .PRINT no matter what the state of the background job. Thus, for urgent messages, .PRINT should be used (rather than .TTYIN or .TTYOUT).

Errors:

None.

Example:

```
          .MCALL  ,,V2,,,REGDEF,,PRINT,,EXIT
          .V2..
          .REGDEF

START:
          .PRINT  #S2
          .PRINT  #S1

          .EXIT

S1:      .ASCIZ  /THIS WILL HAVE CR-LF FOLLOWING/
S2:      .ASCII  /THIS WILL NOT HAVE CR-LF/
          .BYTE   200
          .EVEN

          .END    START
```



## Programmed Requests

### .PROTECT

#### 9.4.25 .PROTECT

The .PROTECT request is used by a job to obtain exclusive control of a vector (two words) in the region 0-476. If it is successful, it indicates that the locations are not currently in use by another job or by the monitor, in which case the job may place an interrupt address and priority into the protected locations and begin using the associated device.

Macro Call: .PROTECT .area, .addr

where: .addr is the address of the word pair to be protected. .addr must be a multiple of four, and must be less than 476 (octal). The two words at .addr and .addr+2 will be protected.

Request Format:

R0 ⇒ .area: 

|       |   |
|-------|---|
| 31    | 0 |
| .addr |   |

Errors:

| <u>Code</u> | <u>Explanation</u>                               |
|-------------|--|
| 0           | Protect failure; locations already in use.       |
| 1           | Address greater than 476 or not a multiple of 4. |

Example:

This example shows the use of .PROTECT to gain control of the UDC11 vectors.

```

.MCALL  ..V2...REGDEF,.PROTECT,.PRINT,.EXIT
..V2..
.REGDEF
STI     MOV      #AREA, -(SP)
        MOV      #234, R5          ;UDC VECTOR ADDRESS
        .PROTECT (SP), R5          ;PROTECT 234, 236
        BCB      ERR              ;YOU CAN'T
        MOV      #UDCINT, (R5)+    ;INITIALIZE THE VECTORS,
        MOV      #340, (R5)        ;AT LEVEL 7

ERR:     .EXIT
        .PRINT   #NOVEC
        .EXIT
AREA:    .BLKW   5
NOVEC:   .ASCIZ  /VECTORS ALREADY IN USE/

```



## Programmed Requests

```
                .EVEN
UDCINT:
:
:
:
```

**.PURGE**

### 9.4.26 .PURGE

The .PURGE request is used to de-activate a channel without performing a .HRESET, .SRESET, .SAVSTATUS, or .CLOSE request. It merely frees a channel without taking any other action. If a tentative file has been .ENTERed on the channel, it will be discarded. Purging an inactive channel acts as a no-op.

Macro Call: .PURGE .chan

Errors:

None.

Example:

The following code is used to make certain that channels 0-7 are free:

```
                .MCALL ..V2... ,REGDEF ,.PURGE ,.EXIT
                ..V2..
                .REGDEF

START:
181 CLR      R1           ;START WITH CHANNEL 0
    .PURGE  R1           ;PURGE A CHANNEL
    INC     R1           ;BUMP TO NEXT CHANNEL
    CMP     R1,#8.       ;IS IT AT CHANNEL 8 YET?
    BLO     15           ;NO, KEEP GOING

    .EXIT
    .END      START
```

**.QSET**

### 9.4.27 .QSET

All RT-11 I/O transfers are done through a centralized queue management system. If I/O traffic is very heavy and not enough queue elements are available, the program issuing the I/O requests may be



## Programmed Requests

suspended until a queue element becomes available. In a F/B system, the other job runs while the first program waits for the element.

The .QSET request is used to make the RT-11 I/O queue larger (i.e., add available entries to the queue). A general rule to follow is that each program should contain one more queue element than the total number of I/O requests which will be active simultaneously. Timing requests such as .TWAIT and .MRKT also cause elements to be used and must be included when allocating queue elements for a program. Note that if synchronous I/O is done (i.e. .READW/.WRITW, etc.) and no timing requests are done, no additional queue elements need be allocated.

Macro Call: .QSET .addr, .qleng

where: .addr is the address at which the new elements are to start.

.qleng is the number of entries to be added. Each queue entry is seven words long; hence the space set aside for the queue should be .qleng \* 7 words.

Each time .QSET is called, a contiguous area of memory is divided into seven-word segments and is added to the queue for that job. .QSET may be called as many times as required. The queue set up by multiple .QSET requests is a linked list. Thus, .QSET need not be called with strictly contiguous arguments. The space used for the new elements is allocated from the user's program space. Thus, care must be taken so that the program in no way alters the elements once they are set up. The .SRESET and .HRESET requests discard all user-defined queue elements; therefore any .QSETs must be reissued.

Care should also be taken to allocate enough memory for the queue. The elements in the queue are altered by the monitor; if enough space is not allocated, destructive references will occur in an unexpected area of memory.

### Errors:

None.

### Example:

```
.MCALL ..V2...,REGDEF,.QSET,.EXIT
..V2..
.REGDEF

START:
.QSET #Q1,#5           ;ADD 5 ELEMENTS TO THE QUEUE
                        ;STARTING AT Q1
.QSET #Q3,#3           ;AND 3 MORE AT Q3.
.EXIT

Q1:  .BLKW  7*5.        ;FIRST QUEUE AREA (35 DECIMAL WORDS)
Q3:  .BLKW  7*3.        ;SECOND QUEUE AREA (21 DECIMAL WORDS)

.END      START
```

Note that Q1 and Q3 need not have been contiguous.



## Programmed Requests

### .RCTRL0

#### 9.4.28 .RCTRL0

The .RCTRL0 request ensures that the console terminal is able to print. Since CTRL O (↑O) struck while output is directed to the console terminal inhibits the output from printing until either another ↑O is struck or until the program resets the ↑O switch, a program that has a message which must appear at the console can override ↑O struck at the keyboard.

Macro Call: .RCTRL0

Errors:

None.

Example:

In this example, the user program first calls the CSI in general mode, then processes the command. When finished, it returns to the CSI for another command line. To make certain that the prompting "\*" typed by the CSI is not inhibited by a CTRL O in effect from the last operation, terminal output is re-enabled via a .RCTRL0 command prior to the CSI call.

```
      .MCALL  ..V2...REGDEF,.RCTRL0,.CSIGEN,.EXIT
      ..V2..
      .REGDEF

START: .RCTRL0                ;MAKE SURE TT OUTPUT IS
                                ;ENABLED
      .CSIGEN #DSPACE,#DEXT,#0 ;CALL CSI-IT WILL TYPE
                                ;"*"

                                ;PROCESS COMMAND

      JMP     START           ;GET NEXT COMMAND

DEXT:  0                      ;NO DEFAULT EXTENSIONS
      0
      0
      0
      0
DSPACE: .B,+400              ;HANDLER SPACE

      .END     START
```



## Programmed Requests

### .RCVD/.RCVDC/.RCVDW

#### 9.4.29 .RCVD/.RCVDC/.RCVDW (F/B Only)

There are three forms of the receive data request; these are used in conjunction with the .SDAT (Send Data) requests to allow a general data/message transfer system. .RCVD requests can be thought of as .READ requests, where data transfer is not from a peripheral device but from the other job in the system.

#### .RCVD

This request is used to receive data and continue execution. The request is posted and the issuing job continues execution. At some point when the job needs to have the transmitted message, an .MWAIT should be executed. This causes the job to be suspended until the message has been received.

Macro Call: .RCVD .area, .buff, .wcnt

where: .buff is the address of the buffer to which the message is to be sent.

.wcnt is the number of words to be transferred.

#### Request Format:

|             |          |   |
|-------------|----------|---|
| R0 ⇒ .area: | 26       | 0 |
|             | (unused) |   |
|             | .buff    |   |
|             | .wcnt    |   |
|             | 1        |   |

Word 0 (the first word) of the message buffer will contain the number of words transmitted whenever the .RCVD is complete. Thus, the space allocated for the message should always be at least one word larger than the actual message size expected.

The word count is a variable number, and as such, the .SDAT/.RCVD combination can be used to transmit a few words or entire buffers. The .RCVD operation is only complete when a .SDAT is issued from the other job.

Programs using .RCVD/.SDAT must be carefully designed to either always transmit/receive data in a fixed format or have the capability of handling variable formats. The messages are all processed in FIFO (first in-first out) order. Thus, the receiver must be certain it is receiving the message it actually wants.



## Programmed Requests

### Errors:

| <u>Code</u> | <u>Explanation</u>                 |
|-------------|------------------------------------|
| 0           | No other job exists in the system. |

### Example:

An example follows the .RCVDW section.

### .RCVDC

The .RCVDC request receives data and enters a completion routine when the message is received. The .RCVDC request is posted and program execution stays with the issuing job. When the other job sends a message, the completion routine specified will be entered.

Macro Call: .RCVDC .area, .buff, . wcnt, .crtn

where: .buff is the address of the buffer to which the message is to be sent.  
.wcnt is the number of words to be transmitted.  
.crtn is the completion routine to be entered.

As in the others, word 0 of the buffer contains the number of words transmitted when the transfer is complete.

### Request Format:

|            |          |   |
|------------|----------|---|
| R0 → area: | 26       | 0 |
|            | (unused) |   |
|            | .buff    |   |
|            | .wcnt    |   |
|            | .crtn    |   |

### Errors:

| <u>Code</u> | <u>Explanation</u>                 |
|-------------|------------------------------------|
| 0           | No other job exists in the system. |

### Example:

An example follows the .RCVDW section.

### .RCVDW

.RCVDW is used to receive data and wait. A message request is posted and the job issuing the request is suspended until the other job sends a message to the issuing job. When the issuing job runs again, the message has been received, and word 0 of the buffer indicates the number of words which were transmitted.



## Programmed Requests

Macro Call: .RCVDW .area, .buff, .wcnt

where: .buff is the address of the buffer to which the message is to be sent.

.wcnt is the number of words to be transmitted.

### Request Format:

|             |          |   |
|-------------|----------|---|
| R0 ⇒ .area: | 26       | 0 |
|             | (unused) |   |
|             | .buff    |   |
|             | .wcnt    |   |
|             | 0        |   |

### Errors:

| Code | Explanation                        |
|------|------------------------------------|
| 0    | No other job exists in the system. |

### Example:

In this example, the running job receives a message from the second job and interprets it as the device and filename of a file to be opened and used. In this case, the message was in RAD50 format, and the receiving program did not use the transmitted length for any purpose.

```

.MCALL ..V2,,,REGDEF,.RCVDW,.PURGE,.LOOKUP,.EXIT,.PRINT
..V2..
.REGDEF

START:
MOV      #AREA,R5          ;R5=EMT ARG. AREA
.RCVDW   R5,#FILE,#4       ;REQUEST MESSAGE AND WAIT
BCS     MERR              ;AN ERROR?
.PURGE   #0                ;CLEAR CHANNEL 0
.LOOKUP  R5,#0,#FILE+2     ;LOOKUP INDICATED FILE
BCS     LKERR             ;ERROR

.EXIT

AREA:    .BLKW  10          ;LEAVE SPACE FOR SAFETY
FILE:    .BLKW  1           ;ACTUAL WORD COUNT IS HERE
         .BLKW  4           ;DEV:FILE,EXT ARE HERE

MERR:    .PRINT  #MMMSG
.EXIT
LKERR:   .PRINT  #LKMSG
.EXIT
MMMSG:   .ASCIZ  /MESSAGE ERROR/
LKMSG:   .ASCIZ  /LOOKUP ERROR/
.EVEN
.END     START

```

The issuing job is suspended until the indicated data is transmitted. Either of the other modes could have also been used to receive the message.



## Programmed Requests

### .READ/.READC/.READW

#### 9.4.30 .READ/.READC/.READW

RT-11 provides three modes of I/O: .READ/.WRITE, .READC/.WRITC, and .READW/.WRITW. Section 9.4.47 explains the output operations. The input operations are described next.

#### .READ

The .READ request transfers a specified number of words from the specified channel to memory. Control returns to the user program immediately after the .READ is initiated. No special action is taken when the transfer is completed.

Macro Call: .READ .area, .chan, .buff, .wcnt, .blk

where: .buff is the address of the buffer to receive the data read.

.wcnt is the number of words to be read.

.blk is the block number to be read relative to the start of the file, not block 0 of the device. The monitor translates the block supplied into an absolute device block number. The user program normally updates .blk before it is used again. If .blk=0, TT: gives ↑ prompt and LP: gives form feed. (This is true for all .READ and .WRITE requests.)

#### Request Format:

|             |       |       |
|-------------|-------|-------|
| R0 → .area: | 10    | .chan |
|             | .blk  |       |
|             | .buff |       |
|             | .wcnt |       |
|             | 1     |       |

When the user program needs to access the data read on the specified channel, a .WAIT request should be issued. This ensures that the data has been read completely. If an error occurred during the transfer, the .WAIT request indicates the error.

#### Note:

See note under .READW request.



## Programmed Requests

### Errors:

| <u>Code</u> | <u>Explanation</u>               |
|-------------|----------------------------------|
| 0           | Attempt to read past end-of-file |
| 1           | Hard error occurred on channel   |
| 2           | Channel is not open              |

### Example:

Refer to the .WRITE/.WRITC/.WRITW examples.

### .READC

The .READC request transfers a specified number of words from the indicated channel to memory. Control returns to the user program immediately after the .READC is initiated. Execution of the user program continues until the .READC is complete, then control passes to the routine specified in the request. When an RTS PC is executed in the completion routine, control returns to the user program.

Macro Call: .READC .area, .chan, .buff, .wcnt, .crtn, .blk

where: .buff is the address of the buffer to receive the data read.

.wcnt is the number of words to be read.

.crtn is the address of the user's completion routine.

.blk is the block number relative to the start of the file, not block 0 of the device. The monitor translates the block supplied into an absolute device block number. The user program normally updates .blk before it is used again.

### Request Format:

R0 ⇒ .area:

|    |                               |
|----|-------------------------------|
| 10 | .chan                         |
|    | .blk                          |
|    | .buff                         |
|    | .wcnt                         |
|    | address of completion routine |

When entering a .READC completion function the following are true:

1. R0 contains the channel status word for the operation. If bit 0 of R0 is set, a hardware error occurred during the transfer. The data may not be reliable.
2. R1 contains the octal channel number of the operation. This is useful when the same completion function is to be used for several different transfers.



## Programmed Requests

### Errors:

| <u>Code</u> | <u>Explanation</u>               |
|-------------|----------------------------------|
| 0           | Attempt to read past end-of-file |
| 1           | Hard error occurred on channel   |
| 2           | Channel is not open              |

### Note:

See note under .READW request.

### Example:

Refer to the .WRITE/.WRITC/.WRITW examples.

### .READW

The .READW request transfers a specified number of words from the indicated channel to memory. Control returns to the user program when the .READW is complete or if an error is detected.

Macro Call: .READW .area, .chan, .buff, .wcnt, .blk

where: .buff is the address of the buffer to receive the data read.

.wcnt is the number of words to be read. The number must be positive.

.blk is the block number relative to the start of the file, not block 0 of the device. The monitor translates the block supplied into an absolute device block number. The user program normally updates .blk before it is used again.

### Request Format:

|             |       |       |
|-------------|-------|-------|
| R0 ⇒ .area: | 10    | .chan |
|             | .blk  |       |
|             | .buff |       |
|             | .wcnt |       |
|             | 0     |       |

On return from this call, the C bit set indicates a hardware error has occurred. If no error occurred, the data is in memory at the specified address. In an F/B system, the other job can be run while the issuing job is waiting for the I/O to complete.

### Note:

Upon return from any of the .READ EMTs, R0 contains the actual number of words read. This will be less than the requested word count if an attempt is made to read past end-of-file, but a partial transfer is possible. No error is returned in this case. Therefore, a program should always use the returned word count as the number of words available. For example, suppose a file is 5 blocks long (i.e., it has block numbers 0 to 4) and a request is issued to read 512 words,



## Programmed Requests

starting at block 4. The request is shortened to 256 words; no error is indicated. This does not apply to non-file structured input, since it is not possible to know how much data there is to be read. Also note that since the request will be shortened to an exact number of blocks, a request for 256 words will either succeed or fail, but cannot be shortened.

### Errors:

| <u>Code</u> | <u>Explanation</u>               |
|-------------|----------------------------------|
| 0           | Attempt to read past end-of-file |
| 1           | Hard error occurred on channel   |
| 2           | Channel is not open              |

### Example:

Refer to the .WRITE/.WRITC/.WRITW examples.

## .RELEASES

### 9.4.31 .RELEASES

The .RELEASES request removes the handler for the specified device from memory. The .RELEASES is ignored if the handler is:

1. Part of RMON (i.e., the system device),
  2. Not currently resident, or
  3. Resident because of a .LOAD command to the Keyboard Monitor,
- .RELEASES from the foreground is always ignored, since the foreground can only use handlers which have been .LOADED.

Macro Call: .RELEASES .devname

where: .devname is the pointer to the .RAD50 device name.

### Errors:

| <u>Code</u> | <u>Explanation</u>        |
|-------------|---------------------------|
| 0           | Handler name was illegal. |



## Programmed Requests

### Example:

In the following example, the DECTape handler (DT) is loaded into memory, used, then released. If the system device is DECTape, the handler is already resident, and .FETCH will return HSPACE in R0.

```

.MCALL ..V2,,,REGDEF,.FETCH,.RELEAS,.EXIT
..V2..
,REGDEF

START: .FETCH #HSPACE,#DTNAME ;LOAD DT HANDLER
      BCS      FERR          ;NOT AVAILABLE

; USE HANDLER

      ,RELEAS #DTNAME          ;MARK DT NO LONGER IN
                                ;MEMORY.

      BR      START

FERR:  HALT

DTNAME: .RAD50 /DT /          ;DT NOT AVAILABLE
HSPACE:                                ;NAME FOR DT HANDLER
                                ;BEGINNING OF HANDLER
                                ;AREA

      ,END      START

```

.RENAME

### 9.4.32 .RENAME

The .RENAME request causes an immediate change of name of the file specified. An error occurs if the channel specified is not already assigned.

Macro Call: .RENAME .area, .chan, .dbl

Request Format:

R0 ⇒ .area: 

|      |       |
|------|-------|
| 4    | .chan |
| .dbl |       |

The argument consists of two consecutive .RAD50 strings. For example:

```

      .RENAME    #AREA,#7,#DBLK    ;USE CHANNEL 7
      BCS        RNMERR            ;NOT FOUND
      .
      .
      .
DBLK:  .RAD50    /DT3/
      .RAD50    /OLDFIL/
      .RAD50    /MAC/
      .RAD50    /DT3/
      .RAD50    /NEWFIL/
      .RAD50    /MAC/

```



## Programmed Requests

The first string represents the file to be renamed and the device it is found on. The second represents the new file name. If a file with the same name as the new file name specified already exists on the indicated device, it is deleted. The second occurrence of the device name DT3 is necessary for proper operation, and should not be omitted. The specified channel is left inactive when the .RENAME is complete. .RENAME requires that the handler to be used be resident at the time the .RENAME request is made. If it is not, a monitor error occurs. Note that .RENAME is legal only on files which are on disk or DECTape. (.RENAMES to other devices are ignored.)

## Errors:

| <u>Code</u> | <u>Explanation</u> |
|-------------|--------------------|
| 0           | Channel open       |
| 1           | File not found     |

## Example:

In the following example, the file DATA.TMP on DT0 is renamed to DATA.001:

```

.MCALL ..V2,..,REGDEF,.FETCH,.PRINT
.MCALL .EXIT,.RENAME
..V2..
.REGDEF

START: .FETCH #HSPACE,#NAMBLK /GET HANDLER
      BCS FERR /SOME ERROR
      .RENAME #AREA,#0,#NAMBLK /DO THE RENAME
      BCS RNMERR /ERROR
      .EXIT
FERR: .PRINT #FMSG
      .EXIT
RNMERR: .PRINT #RMSG
      .EXIT
AREA: .BLKW 5 /ROOM FOR ARGS.
NAMBLK: .RAD50 /DT0DATA TMP/ /OLD NAME
      .RAD50 /DT0DATA 001/ /NEW NAME
FMSG: .ASCIZ /FETCH?/ /ERROR MESSAGES
RMSG: .ASCIZ /RENAME?/
      .EVEN
HSPACE:
      .END START

```



## Programmed Requests

### .REOPEN

#### 9.4.33 .REOPEN

The .REOPEN request reassociates the specified channel with a file on which a .SAVESTATUS was performed. The .SAVESTATUS/.REOPEN combination is useful when a large number of files must be operated on at one time. As many files as are needed can be opened with .LOOKUP, and their status preserved with .SAVESTATUS. When data is required from a file, a .REOPEN enables the program to read from the file. The .REOPEN need not be done on the same channel as the original .LOOKUP and .SAVESTATUS.

Macro Call: .REOPEN .area, .chan, .cblk

where: .cblk is the address of the five-word block where the channel status information was stored.

Request Format:

R0  $\Rightarrow$  .area: 

|       |       |
|-------|-------|
| 6     | .chan |
| .cblk |       |

Errors:

| <u>Code</u> | <u>Explanation</u>  |
|-------------|---|
| 0           | The specified channel is in use. The .REOPEN has not been done. |

Example:

Refer to the example following the description of .SAVESTATUS.

### .SAVESTATUS

#### 9.4.34 .SAVESTATUS

The .SAVESTATUS request stores five data words into a user-specified area of memory. These words contain all the information RT-11



## Programmed Requests

requires to completely define a file. When a .SAVESTATUS is done, the data words are placed in memory, and the specified channel is again available for use. When the saved channel data is required, the .REOPEN request is used.

.SAVESTATUS can only be used if a file has been opened with .LOOKUP. If .ENTER was used, .SAVESTATUS is illegal and returns an error. Note that .SAVESTATUS is legal only on files which are not on magtape or cassette.

Macro Call: .SAVESTATUS .area .chan, .cblk

where: .cblk is the address of the user memory block (5 words) where the channel status information is to be stored.

### Request Format:

R0 → .area: 

|       |       |
|-------|-------|
| 5     | .chan |
| .cblk |       |

The five words stored are the five words normally contained in the channel area, as follows:

| <u>Word #</u> | <u>Contents</u>   |              |                 |   |  |     |   |   |  |   |  |      |  |    |  |    |         |    |  |
|---------------|---|--------------|-----------------|---|--|-----|---|---|--|---|--|------|--|----|--|----|---------|----|--|
| 1             | Channel status word. The contents of the bits of this word are: <table><tr><th><u>Bit #</u></th><th><u>Contents</u></th></tr><tr><td>0</td><td>1 - a hardware error occurred on this channel.</td></tr><tr><td>1-5</td><td>Index into monitor tables. This describes the physical device with which the channel is associated.</td></tr><tr><td>6</td><td>1 - a .RENAME operation is in progress on the channel.</td></tr><tr><td>7</td><td>1 - a .CLOSE operation must rewrite the directory (i.e., set when a .ENTER is done).</td></tr><tr><td>8-12</td><td>Contains the directory segment number (1-37(8)) in which the current open file can be found.</td></tr><tr><td>13</td><td>1 - An end-of-file was found on the channel.</td></tr><tr><td>14</td><td>Unused.</td></tr><tr><td>15</td><td>1 - This channel is currently in use (i.e., a file is open on this channel).</td></tr></table> | <u>Bit #</u> | <u>Contents</u> | 0 | 1 - a hardware error occurred on this channel. | 1-5 | Index into monitor tables. This describes the physical device with which the channel is associated. | 6 | 1 - a .RENAME operation is in progress on the channel. | 7 | 1 - a .CLOSE operation must rewrite the directory (i.e., set when a .ENTER is done). | 8-12 | Contains the directory segment number (1-37(8)) in which the current open file can be found. | 13 | 1 - An end-of-file was found on the channel. | 14 | Unused. | 15 | 1 - This channel is currently in use (i.e., a file is open on this channel). |
| <u>Bit #</u>  | <u>Contents</u>   |              |                 |   |  |     |   |   |  |   |  |      |  |    |  |    |         |    |  |
| 0             | 1 - a hardware error occurred on this channel.  |              |                 |   |  |     |   |   |  |   |  |      |  |    |  |    |         |    |  |
| 1-5           | Index into monitor tables. This describes the physical device with which the channel is associated.   |              |                 |   |  |     |   |   |  |   |  |      |  |    |  |    |         |    |  |
| 6             | 1 - a .RENAME operation is in progress on the channel.  |              |                 |   |  |     |   |   |  |   |  |      |  |    |  |    |         |    |  |
| 7             | 1 - a .CLOSE operation must rewrite the directory (i.e., set when a .ENTER is done).  |              |                 |   |  |     |   |   |  |   |  |      |  |    |  |    |         |    |  |
| 8-12          | Contains the directory segment number (1-37(8)) in which the current open file can be found.  |              |                 |   |  |     |   |   |  |   |  |      |  |    |  |    |         |    |  |
| 13            | 1 - An end-of-file was found on the channel.  |              |                 |   |  |     |   |   |  |   |  |      |  |    |  |    |         |    |  |
| 14            | Unused.   |              |                 |   |  |     |   |   |  |   |  |      |  |    |  |    |         |    |  |
| 15            | 1 - This channel is currently in use (i.e., a file is open on this channel).  |              |                 |   |  |     |   |   |  |   |  |      |  |    |  |    |         |    |  |
| 2             | Starting block number of the file. Zero for non-file structured devices.  |              |                 |   |  |     |   |   |  |   |  |      |  |    |  |    |         |    |  |
| 3             | Length of file (in 256-word blocks).  |              |                 |   |  |     |   |   |  |   |  |      |  |    |  |    |         |    |  |
| 4             | Data length of file; currently unused.  |              |                 |   |  |     |   |   |  |   |  |      |  |    |  |    |         |    |  |
| 5             | Even Byte: I/O count. Count of how many I/O requests have been made on this channel. Odd Byte:  |              |                 |   |  |     |   |   |  |   |  |      |  |    |  |    |         |    |  |



## Programmed Requests

unit number of the device associated with the channel  
(between 0 - 7).

While the .SAVESTATUS/.REOPEN combination is very useful, care must be observed when using it. In particular, the following cases should be avoided:

1. If a .SAVESTATUS is performed and the same file is then deleted before it is reopened, it becomes available as an empty space which could be used by the .ENTER command. If this sequence occurs, the contents of the file supposedly saved will change.
2. Although the device handler for the required peripheral need not be in memory for execution of a .REOPEN, if the handler is not in memory when a .READ or .WRITE is executed, a fatal error is generated.

### Errors:

| <u>Code</u> | <u>Explanation</u>  |
|-------------|---|
| 0           | The channel specified is not currently associated with any file, i.e. a previous .LOOKUP on the channel was never done. |
| 1           | The file was opened via .ENTER, or is a magtape or cassette file, and a .SAVESTATUS is illegal.                         |

### Example:

One of the more common uses of .SAVESTATUS and .REOPEN is to consolidate all directory access motion and code at one place in the program. All files necessary are opened and their status saved, then they are re-opened one at a time as needed. USR swapping can be minimized by locking in the USR, doing .LOOKUPS as needed, using .SAVESTATUS to save the file data, and then .UNLOCKing the USR.

In the program segment below, three input files are specified in the command string; these are then processed one at a time.

```
.MCALL ..V2,..REGDEF,.CSIGEN,.SAVESTATUS,.REOPEN
.MCALL .READ,.EXIT
..V2..
.REGDEF

START:  MOV      #AREA,R5
        .CSIGEN #DSPACE,#DEXT    ;GET INPUT FILES

        MOV      R0,BUFF          ;SAVE POINTER TO FREE CORE

        .SAVESTATUS R5,#3,#BLOCK1 ;SAVE FIRST INPUT FILE
        .SAVESTATUS R5,#4,#BLOCK2 ;SAVE SECOND FILE
        .SAVESTATUS R5,#5,#BLOCK3 ;SAVE THIRD FILE

        MOV      #BLOCK1,R4
PROCESS: .REOPEN R5,#0,R4          ;REOPEN FILE ON
                                   ;CHANNEL 0

        .READ    R5,#0,BUFF,COUNT,BLOCK ;PROCESS FILE ON CHANNEL 0

DONE:   ADD      #12,R4            ;POINT TO NEXT SAVESTATUS BLOCK
        CMP      R4,#BLOCK3       ;LAST FILE PROCESSED?
                                   9-79
```



## Programmed Requests

```

        BLOS      PROCESS      ;NO = DO NEXT
        .EXIT

BLOCK1: .WORD     0,0,0,0,0      ;MEMORY BLOCKS FOR
BLOCK2: .WORD     0,0,0,0,0      ;SAVESTATUS INFORMATION
BLOCK3: .WORD     0,0,0,0,0
AREA:   .BLKW     10

BUFF:   .WORD     0
BLOCK:  .WORD     0
COUNT: .WORD     256.

DEXT:   .WORD     0,0,0,0
DSPACE: .END      START

```

### .SDAT/.SDATC/.SDATW

#### 9.4.35 .SDAT/.SDATC/.SDATW

These requests are used in conjunction with the .RDVD/.RCVDW/.RCVDC calls to allow message transfers with RT-11. .SDAT transfers can be considered similar to .WRITE requests in which data transfer is not from a peripheral, but from one job to another.

#### .SDAT

Macro Call: .SDAT .area, .buff, .wcnt

where: .buff is the buffer address of the beginning of the message to be transferred.

.wcnt is the number of words to transfer.

#### Request Format:

|             |        |   |
|-------------|--------|---|
| R0 → .area: | 25     | 0 |
|             | unused |   |
|             | .buff  |   |
|             | .wcnt  |   |
|             | 1      |   |

#### Errors:

| Code | Explanation          |
|------|----------------------|
| 0    | No other job exists. |



## Programmed Requests

### Example:

See the example following .SDATW.

### .SDATC

Macro Call: .SDATC .area, .buff, .wcnt, .crtn

where: .buff is the buffer address of the beginning of the message to be transferred.

.wcnt is the number of words to transfer.

.crtn is the address of the completion routine to be entered when the message has been transmitted.

### Request Format:

|             |        |   |
|-------------|--------|---|
| R0 ⇒ .area: | 25     | 0 |
|             | unused |   |
|             | .buff  |   |
|             | .wcnt  |   |
|             | .crtn  |   |

### Errors:

| Code | Explanation          |
|------|----------------------|
| 0    | No other job exists. |

### Example:

See the example following .SDATW.

### .SDATW

Macro Call .SDATW .area, .buff, .wcnt

where: .buff is the buffer address of the beginning of the message to be transferred

.wcnt is the number of words to transfer

### Request Format:

|             |        |   |
|-------------|--------|---|
| R0 ⇒ .area: | 25     | 0 |
|             | unused |   |
|             | .buff  |   |
|             | .wcnt  |   |
|             | 0      |   |



## Programmed Requests

### Errors:

| <u>Code</u> | <u>Explanation</u>   |
|-------------|----------------------|
| 0           | No other job exists. |

### Example:

In this example, the job first sends a message interrogating the other job about the status of an operation, and then looks for an acknowledgement from the job.

```

.MCALL ..V2,,,REGDEF,.SDAT,.RCVD,.MWAIT,.PRINT,.EXIT
..V2..
.REGDEF

START:
MOV      #AREA,R5          ;SET UP EMT BLOCK
.SDAT    R5,#SBUF,#MLGTH  ;ASK HIM A QUESTION
BCS      NOJOB             ;NO OTHER JOB AROUND!

                                ;MISCELLANEOUS PROCESSING

.RCVD     R5,#BUFF2,#20.   ;RECEIVE 20 DECIMAL WORDS
.MWAIT
MOV      #BUFF2+2,R1       ;POINT TO ACTUAL ANSWER.
CMPB     (R1)+,#'Y         ;IS FIRST WORD Y FOR YES?
BNE      PRNEG             ;NEGATIVE ACKNOWLEDGE
.PRINT    #POSACK

.EXIT

PRNEG:   .PRINT #NEGACK     ;NEGATIVE ON OUR INQUIRY
.EXIT
SBUF:    .ASCII /IS THE REQUIRED PROCESS GOING?/
MLGTH:   .SBUF
BUFF2:   .WORD 0            ;ACTUAL LENGTH IS HERE
        .BLKW 20.          ;SPACE FOR 20, WORDS

NOJOB:   .PRINT #NJMSG
.EXIT
NEGACK:  .ASCIZ /NEGATIVE ACKNOWLEDGE/
POSACK:  .ASCIZ /POSITIVE ACKNOWLEDGE/
NJMSG:   .ASCIZ /NO JOB/
        .EVEN
AREA:    .BLKW 10.

.END     START

```

## .SETTOP

### 9.4.36 .SETTOP

The .SETTOP request allows the user program to request that a new address be specified as a program's upper limit. The monitor



## Programmed Requests

determines whether this address is legal and whether or not a memory swap is necessary when the USR is required. For instance, if the program specified an upper limit below the start address of USR, no swapping is necessary, as the USR is not overlaid. If .SETTOP from the background specifies a high limit greater than the address of the USR and a SET USR NOSWAP command has not been given, a memory swap is not required. Section 9.2.5 gives details on determining where the USR is in memory and how to optimize the .SETTOP.

On return from .SETTOP, both R0 and the word at location 50 (octal) contain the highest memory address allocated for use. If the job requested an address higher than the highest address which is legal for the requesting job, it is adjusted down to that address.

Macro Call: .SETTOP .addr

where: .addr is the address of the word immediately following the free area desired.

### Notes:

1. A program should never do a .SETTOP and assume that its new upper limit is the address it requested. It must always examine the returned contents of R0 or location 50 to determine its actual high address.
2. In Version 1 of RT-11, R0 did not return the high address in R0, but only in word 50.
3. It is imperative that the value returned in R0 or location 50 be used as the absolute upper limit. If this value is ever exceeded, vital parts of the monitor may be destroyed, and the system integrity will be violated.

### Errors:

None.

### Example:

Following is an example in two parts. The first indicates how a background job can be assured of reserving space up to but not including the USR. This in effect gives the job all the space it can without causing the USR to become non-resident.

The second part indicates how to always reserve the maximum amount of space by making the USR non-resident.

```
I)      .MCALL  ..V2,,,REGDEF,.SETTOP,.EXIT
          ..V2..
          .REGDEF
```

```
START:
RMON#54
USR#266
```

```
;POINTER TO START OF RESIDENT
;OFFSET FROM RESIDENT TO POINTER
;WHERE USR WILL START.
```



## Programmed Requests

```

MOV    @RMON,R1      ;START OF RMON TO R1
.SETTOP USR(R1)      ;266 BYTES INTO RESIDENT IS A WORD
                    ;WHICH ALWAYS INDICATES WHERE THE
                    ;USR BEGINS.

MOV    R0,HICORE      ;R0 CONTAINS THE HIGH ADDRESS
                    ;THAT WAS RETURNED.

II)      .SETTOP #-2  ;IF WE ASK FOR A VALUE GREATER
                    ;THAN START OF RESIDENT, WE
                    ;WILL GET BACK THE ABSOLUTELY
                    ;HIGHEST USABLE ADDRESS.
                    ;THAT IS OUR LIMIT NOW

MOV    R0,HICORE

.EXIT
HICORE: .WORD 0
.END    START

```

If a SET USR NOSWAP command is executed, the USR cannot be made non-resident. In this case, in both I & II above, R0 would return a value just below the USR.

## .SFPA

### 9.4.37 .SFPA

.SFPA allows users with floating point hardware (FPP on 11/45 and FIS on 11/40) to set trap addresses to be entered when a floating point exception occurs. If no user trap address is specified and a floating point (FP) exception occurs, a ?M-FP TRAP occurs, and the job is aborted.

Macro Call: .SFPA .area, .addr

where: .addr is the address of the routine to be entered when an exception occurs.

Request Format:

R0 ⇒ .area:

|       |   |
|-------|---|
| 30    | 0 |
| .addr |   |

Notes:

1. If the address argument is 0, user floating point routines are disabled and the fatal ?M-FP TRAP error is produced.
2. In the F/B environment, an address value of 1 indicates that the FP registers should be switched when a context switch occurs, but no user traps are enabled. This allows both jobs to use the FP unit. An address of 1 to the Single-Job Monitor is equivalent to an address of 0.



## Programmed Requests

3. When the user routine is activated, it is necessary to re-execute an .SFPA request, as the monitor inhibits user traps when any one is serviced. It does this to inhibit any possible infinite loop being set up by repeated FP exceptions.
4. If the 11/45 FPP is being used, the instruction STST -(SP) is executed by the monitor before entering the user's trap routine. Thus, the trap routine must pop the two status words off the stack before doing an RTI. The program can tell if FPP hardware is available by examining the configuration word in the monitor (see Section 9.2.6).

### Errors:

None.

### Example:

This example sets up a user FP trap address.

```
.MCALL ..V2... ,REGDEF ,.SFPA ,.EXIT
..V2..
,REGDEF

START:

.SFPA #AREA ,#FPTRAP
.EXIT

FPTRAP:

.SFPA #AREA ,#FPTRAP
RTI

AREA: .BLKW 10
.END START
```

.SPFUN

### 9.4.38 .SPFUN

This request is used only in conjunction with cassette and magtape handlers. It provides a means for doing device-dependent functions, such as rewind and backspace, to those devices. If .SPFUN is requested for any device except MT and CT, the request is ignored.

Macro Call: .SPFUN .area, .chan, .code, .buff, .wcnt, .crtn, .blk

where: .code is the numerical code of the function to be performed



## Programmed Requests

`.crtn` is the entry point of a completion routine.  
If left blank, 0 is automatically inserted.

### Request Format:

|             |       |       |
|-------------|-------|-------|
| R0 ⇒ .area: | 32    | .chan |
|             | .blk  |       |
|             | .buff |       |
|             | .wcnt |       |
|             | .code | 377   |
|             | .crtn |       |

All other arguments are the same as those defined for `.READ/.WRITE` requests (Sections 9.4.30 and 9.4.47). They are only required when doing a `.WRITE` with extended record gap to MT. If the `.crtn` argument is left blank, the requested operation will complete before control returns to the user program. `.crtn=1` is equivalent to executing a `.READ` or `.WRITE` in that the function is initiated and returns immediately to the user program. A `.WAIT` on the channel ensures that the operation is completed. If `.crtn=N`, it is taken as a completion routine address to be entered when the operation is complete.

The available functions and their codes are:

| <u>Function</u>       | <u>MT</u> | <u>CT</u> |
|-----------------------|-----------|-----------|
| Forward to last file  |           | 377       |
| Forward to last block |           | 376       |
| Forward to next file  |           | 375       |
| Forward to next block |           | 374       |
| Rewind to load point  | 373       | 373       |
| Write file gap        |           | 372       |
| Write EOF             | 377       |           |
| Forward 1 record      | 376       |           |
| Backspace 1 record    | 375       |           |
| Write with extended   |           |           |
| file gap              | 374       |           |
| Offline               | 372       |           |

To use the `.SPFUN` request, the handler must be in memory and a channel associated with a file via a `.LOOKUP` request.

Refer to Appendix H for details of MT and CT handlers.

### Errors:

Errors are detected in the same way as for the `.READ/.READC/.READW` requests. Refer to Section 9.4.30 for details.

### Example:

The following example rewinds a cassette and writes out a 256-word buffer and then a file gap.

```
.MCALL ..V2,,,REGDEF,.FETCH,.LOOKUP,.SPFUN,.WRITW
.MCALL .EXIT,.PRINT,.WAIT,.CLOSE
..V2..
.REGDEF
```

### START:

```
.FETCH #HSPC,#CT          IGET A HANDLER
```



## Programmed Requests

```

BCS      FERR      /FETCH ERROR
,LOOKUP  #AREA,#4,#CT /LOOK IT UP ON CHANNEL 4
BCS      LKERR      /LOOKUP ERROR
,SPFUN   #AREA,#4,#373 /REWIND SYNCHRONOUSLY
BCS      SPERR      /AN ERROR OCCURRED,
MOV      #3,R5      /COUNT
                        /BLOCK 0,
,WRITH   #AREA,#4,#BUFF,#256.,BLK
BCS      WTERR
,SPFUN   #AREA,#4,#372,,#1 /ASYNCHRONOUS FILE GAP
,PRINT   #DONE
,WAIT    #4          /WAIT FOR DONE
,CLOSE   #4          /CLOSE THE FILE
,EXIT

AREA:    .BLKW      10
FERR:    .PRINT     #FMSG
,EXIT
LKERR:    .PRINT     #LKMSG
,EXIT
SPERR:    .PRINT     #SPMSG
,EXIT
WTERR:    .PRINT     #WTMSG
,EXIT
DONE:    .ASCIZ     /ALL DONE/
FMSG:    .ASCIZ     /FETCH?/
LKMSG:    .ASCIZ     /FILE?/
SPMSG:    .ASCIZ     /SPECIAL FUNCTION ERROR/
WTMSG:    .ASCIZ     /WRITE ERROR/
,EVEN
CT:      .RAD50     /CT /
,WORD    0,0,0
BUFF:    .BLKW      256.
BLK:     .WORD      0
HSPC:    .END       START

```

.SPND/.RSUM

### 9.4.39 .SPND/.RSUM (F/B only)

The .SPND/.RSUM requests allow a job to control execution of its mainstream code (that code which is not executing as a result of a completion routine). .SPND suspends the mainstream and allows only completion routines (for I/O and mark time requests) to run. .RSUM from one of the completion routines resumes the mainstream code. These functions enable a program to wait for a particular I/O or mark time request by suspending the mainstream and having the selected event's completion routine issue a .RSUM. This differs from the .WAIT request, which suspends the mainstream until all I/O operations on a specific channel have completed.



## Programmed Requests

.SPND

Macro Call: .SPND

where: R0 ⇒ 

|   |   |
|---|---|
| 1 | 0 |
|---|---|

.RSUM

Macro Call: .RSUM

where: R0 ⇒ 

|   |   |
|---|---|
| 2 | 0 |
|---|---|

### Notes:

1. The monitor maintains a suspension counter for each job. This counter is decremented by .SPND and incremented by .RSUM. A job will actually be suspended only if this counter is negative. Thus, if a .RSUM is issued before a .SPND, the latter request will return immediately.
2. A program must issue an equal number of .SPNDs and .RSUMs.
3. A .RSUM request from the mainstream code increments the suspension counter.
4. A .SPND request from a completion routine decrements the suspension counter, but does not suspend the mainstream. If a completion routine does a .SPND, the mainstream continues until it also issues a .SPND, at which time it is suspended and will require two .RSUMs to proceed.
5. Since a .TWAIT is simulated in the monitor using suspend and resume, a .RSUM issued from a completion routine may cause the mainstream to continue past a timed wait before the entire time interval has elapsed.

### Errors:

None.

### Example:

In this example, the program starts a number of read operations and suspends itself until at least two of them are complete.

```
.MCALL ..V2,,,REGDEF,.SPND,.RSUM,.READC,.EXIT,.LOOKUP
.MCALL .PRINT,.WAIT
..V2,.
.REGDEF
```

### START:

```
.LOOKUP #AREA,#2,#FILE2
BCS 15
.LOOKUP #AREA,#3,#FILE3
BCS 15
.LOOKUP #AREA,#4,#FILE4
BCC 35
```



# Programmed Requests

```

181 .PRINT #23
    .EXIT
291 .ASCIZ /LOOKUP ERROR/
    .EVEN
381

```

```

MOV #2,RSVCTR ;WAIT FOR 2 COMPLETIONS
MOV #AREA,R5
.READC R5,#2,#BUF1,COUNT1,#CROUTN,BLOK1
BCS ERROR
.READC R5,#3,#BUF2,COUNT2,#CROUTN,BLOK2
BCS ERROR
.READC R5,#4,#BUF3,COUNT3,#CROUTN,BLOK3
BCS ERROR
.SPND

```

```

.WAIT #2
.WAIT #3
.WAIT #4
.EXIT

```

```

CROUTN: ASL R1 ;DOUBLE CHANNEL # FOR INDEXING
        INC DONEFL(R1) ;R1=CHANNEL THAT IS DONE
        ROR R0 ;SET A FLAG SAYING 80.
        ADC ERRFLG(R1) ;ANY ERRORS?
        DEC RSVCTR ;IF CARRY SET, SET ERROR FLAG FOR CHANNEL
        BNE 18 ;ARE WE THE SECOND TO FINISH?
        .RSUM ;NO
181 RTS PC ;YES, START UP

```

```

ERROR: .PRINT #RDMMSG
        .EXIT
RDMMSG: .ASCIZ /READ ERROR/
        .EVEN
AREA: .BLKW 10
RSVCTR: .WORD 0
COUNT1: .WORD 256.
COUNT2: .WORD 256.
COUNT3: .WORD 256.
BLOK1: .WORD 0
BLOK2: .WORD 0
BLOK3: .WORD 0
FILE2: .RAD50 /DK TEST2 TMP/
FILE3: .RAD50 /DK TEST3 TMP/
FILE4: .RAD50 /DK TEST4 TMP/
DONEFL: .WORD 0,0,0
ERRFLG: .WORD 0,0,0
BUF1: .BLKW 256.
BUF2: .BLKW 256.
BUF3: .BLKW 256.

.END START

```



## Programmed Requests

### **.SRESET**

#### 9.4.40 .SRESET

The .SRESET (software reset) request performs the following functions:

1. Dismisses any device handlers which were brought into memory via a .FETCH call. Handlers which were loaded via the Keyboard Monitor LOAD command remain resident, as does the system device handler.
2. Purges any currently open files. Files opened for output with .ENTER will never be made permanent.
3. Reverts to using only 16 (decimal) I/O channels. Any channels defined with .CDFN are disregarded. A .CDFN must be reissued to open more than 16 (decimal) channels after a .SRESET is performed.
4. Resets the I/O queue to one element.
5. Initializes locations 40-57.

Macro Call .SRESET

Errors:

None.

Example:

In the example below, .SRESET is used prior to calling the CSI to ensure that all handlers are removed from memory and the CSI is started with a free handler area.

```
.MCALL ..V2...REGDEF,.CSIGEN,.SRESET,.EXIT
..V2..
.REGDEF
```

```
START: .CSIGEN #DSPACE,#DEXT,#0 ;GET COMMAND STRING
MOV      R0,BUFFER           ;R0 POINTS TO FREE MEMORY
```

```
DONE:  .SRESET                ;RELEASE HANDLERS, DELETE
                                ;TENTATIVE FILES
BR      START                ;AND REPEAT PROGRAM,
```



## Programmed Requests

```
DEXT: .WORD 0,0,0,0      ;NO DEFAULT EXTENSIONS
BUFFER: 0
DSpace: , ;START OF HANDLER AREA.

.END START
```

If the .SRESET had not been performed prior to the second call of .CSIGEN, it is possible that the second command string would load a handler over one that the monitor thought was resident from the first command line.

**.TLOCK**

### 9.4.41 .TLOCK

.TLOCK is used in an F/B system to attempt to gain ownership of the USR. It is similar to .LOCK in that if successful, the user job returns with the USR in memory. However, if a job attempts to .LOCK the USR while the other job is using it, the requesting job is suspended until the USR is free. With .TLOCK, if the USR is not available, control returns immediately with the C bit set to indicate the .LOCK request failed.

Macro Call: .TLOCK

The .TLOCK request allows the job to continue running, with only one sub-job affected. With a .LOCK request, all sub-jobs would be automatically suspended, and the other job in the system would run.

Request Format:

R0: ⇒ area: 

|   |   |
|---|---|
| 7 | 0 |
|---|---|

Errors:

| <u>Code</u> | <u>Explanation</u>                    |
|-------------|---------------------------------------|
| 0           | USR is already in use by another job. |

Example:

In the following example, the user program needs the USR for a sub-job it is executing. If it fails to get the USR it suspends that sub-job and runs another sub-job. This type of procedure is useful to schedule several sub-jobs within a background or foreground program.

```
.MCALL ..V2,,,REGDEF,.TLOCK,.LOOKUP,.UNLOCK,.EXIT,.PRINT
..V2..
.REGDEF
```

START:



## Programmed Requests

```

.TLOCK                                ;GET THE USR
BCS      SUSPND                       ;FAILED, SUSPEND SUB-JOB
.LOOKUP #AREA,#4,#JINAM ;LOOKUP A FILE
BCS      LKERR
.UNLOCK                                ;RELEASE USR

.EXIT

SUSPND: JSR      PC,SP8JOB             ;SUSPEND SUB-JOB
        JSR      PC,SCHED             ;AND SCHEDULE NEXT USER

AREA:   .BLKW    10
JINAM:  .RAD50   /DK TEST1 TMP/
LKERR:  .PRINT   #LKMSG
        .EXIT
LKMSG:  .ASCIZ   /LOOKUP ERROR/
        .EVEN
SP8JOB: RTS      PC
SCHED:  RTS      PC

.END      START

```

## .TRPSET

### 9.4.42 .TRPSET

.TRPSET allows the user job to intercept traps to 4 and 10 instead of having the job aborted with a ?M-TRAP TO 4 or ?M-TRAP TO 10 message. If .TRPSET is in effect when a trap occurs, the user-specified routine is entered. The sense of the C bit on entry to the routine determines which trap occurred: C bit clear indicates a trap to 4; set indicates a trap to 10. The user routine should exit via an RTI instruction.

Macro Call: .TRPSET .area, .addr

where: .addr is the address of the user's trap routine.  
If an address of 0 is specified, the user's trap interception is disabled.

Request Format:

|             |       |   |
|-------------|-------|---|
| R0 ⇒ .area: | 3     | 0 |
|             | .addr |   |

Notes:

It is necessary to reissue a .TRPSET request whenever a trap occurs and the user routine is entered. The monitor inhibits servicing user traps prior to entering the first user trap routine. Thus, if a trap should occur from within the user's trap routine, a ?M-TRAP message is generated. The last operation the user routine should perform before an RTI is to reissue the .TRPSET request.



## Programmed Requests

Errors:

None.

Example:

The following example sets up a user trap routine and, when the trap occurs, prints an appropriate error message.

```
      .MCALL  ..V2...REGDEF,.TRPSET,.EXIT,.PRINT
      ..V2..
      .REGDEF

START:

      .TRPSET #AREA,#TRPLOC
      MOV     #101,R0          ;SET TO PRODUCE A TRAP
      TST     (R0)+            ;THIS WILL TRAP TO 4.
      .WORD   67              ;THIS WILL TRAP TO 10.
      .EXIT

TRPLOC: BCS     18             ;C SET = TRAP TO 10.
      .PRINT  #TRP4          ;TRAP TO 4
      BR      25
18:    .PRINT  #TRP10         ;TRAP TO 10
25:    .TRPSET #AREA,#TRPLOC  ;RESET TRAP ADDRESS
      RTI

AREA1   .BLKW   10
TRP4:   .ASCIZ  /TRAP TO 4/
TRP10:  .ASCIZ  /TRAP TO 10/
      .EVEN

      .END    START
```

.TTYIN/.TTINR

### 9.4.43 .TTYIN/TTINR

These requests are used to transfer characters from the console terminal to the user program. The character thus obtained appears right-justified (even byte) in R0.



## Programmed Requests

The expansion of .TTYIN is:

EMT 340  
BCS .-2

while that for .TTINR is:

EMT 340

If no characters or lines are available when an EMT 340 is executed, return is made with the C bit set. The implication of these calls is that .TTYIN causes a tight loop waiting for a character/line to appear, while the user can either wait or continue processing using .TTINR.

Macro Calls: .TTYIN .char

.TTINR

where: .char is the location where the character in R0 is stored. If not specified, the character is left in R0.

If the carry bit is set when execution of the .TTINR request is completed, it indicates that no character was available; the user has not yet typed a valid line. Under the F/B Monitor, .TTINR does not return the carry bit set unless bit 6 of the Job Status Word was on when the request was issued (see below).

There are two modes of doing console terminal input. This is governed by bit 12 of the Job Status Word. If bit 12 = 0, normal I/O is performed. In this mode, the following conditions apply:

1. The monitor echoes all characters typed; lower case characters are converted to upper case.
2. CTRL U (↑U) and RUBOUT perform line deletion and character deletion, respectively.
3. A carriage return, line feed, CTRL Z, or CTRL C must be struck before characters on the current line are available to the program. When carriage return is typed, characters on the line typed are passed one-by-one to the user program. Both carriage return and line feed are passed to the program.
4. ALTMODES (octal codes 175 and 176) are converted to ESCAPES (octal 33).

If bit 12 = 1, the console is in special mode. The effects are:

1. The monitor does not echo characters typed except for CTRL C and CTRL O.
2. CTRL U and RUBOUT do not perform special functions.
3. Characters are immediately available to the program.
4. No ALTMODE conversion is done.

In special mode, the user program must echo the characters received. However, CTRL C and CTRL O are acted on by the monitor in the usual



## Programmed Requests

way. Bit 12 in the JSW must be set by the user program. This bit is cleared when control returns to RT-11.

CTRL F and CTRL B are not affected by the setting of bit 12. The monitor always acts on these characters.

Under the F/B Monitor, if a terminal input request is made and no character is available, job execution is blocked until a character is ready. This is true for both .TTYIN and .TTINR, and for both normal and special modes. If a program really requires execution to continue and the carry bit to be returned, it must turn on bit 6 of the JSW (location 44) before the .TTINR request. Bit 6 is cleared when a program terminates.

### Errors:

| <u>Code</u> | <u>Explanation</u>                      |
|-------------|---|
| 0           | No characters available in ring buffer. |

### Example:

Refer to the example following the description of .TTYOUT/.TTOUTR.

**.TTYOUT/.TTOUTR**

#### 9.4.44 .TTYOUT/.TTOUTR

These requests cause a character to be transmitted from R0 to the console terminal. The difference, as in the .TTYIN/.TTINR requests, is that if there is no room for the character in the monitor's buffer, the .TTYOUT request waits for room before proceeding, while the .TTOUTR does not wait for room and the character in R0 is not output.

Macro Calls: .TTYOUT .char  
                  .TTOUTR

where: .char       is the location containing the character to be loaded in R0 and printed. If not specified, the character in R0 is printed. Upon return from the request, R0 still contains the character.

If the carry bit is set when execution of the .TTOUTR request is completed, it indicates that there is no room in the buffer and that no character was output. Under the F/B Monitor, .TTOUTR normally does not return the carry bit set. Instead, the job is blocked until room



## Programmed Requests

is available in the output buffer. If a job really requires execution to continue and the carry bit to be returned, it must turn on bit 6 of the Job Status Word (location 44) before issuing the request.

The .TTINR and .TTOUTR requests have been supplied as a help to those users who do not wish to suspend program execution until a console operation is complete. With these modes of I/O, if a no-character or no-room condition occurs, the user program can continue processing and try the operation again at a later time.

### Note:

If a foreground job leaves bit 6 on in the JSW, any further foreground .TTYIN or .TTYOUT requests will cause the system to lock out the background. Note also that each partition has its own JSW, and therefore can be in different terminal modes independently.

### Errors:

| <u>Code</u> | <u>Explanation</u>       |
|-------------|--------------------------|
| 0           | Output ring buffer full. |

### Example:

As an example of the various terminal requests, the following program is coded in two ways. The program itself accepts a line from the keyboard, then repeats it on the terminal.

The first example uses .TTYIN and .TTYOUT, which are synchronous. The monitor retains control until both requests are satisfied, hence there is no time available for any other processing while waiting.

```
      .MCALL  ..V2...REGDEF,.TTYIN,.TTYOUT
      ..V2..
      .REGDEF

START:  MOV    #BUFFER,R1      ;POINT R1 TO BUFFER
        CLR    R2              ;CLEAR CHARACTER COUNT
INLOOP: .TTYIN  (R1)+          ;READ CHAR INTO BUFFER
        INC    R2              ;BUMP COUNT
        CMPB   #12,R0          ;WAS LAST CHAR=LF?
        BNE    INLOOP         ;NO-GET NEXT
        MOV    #BUFFER,R1      ;YES-POINT R1 TO BUFFER
OUTLOOP: .TTYOUT (R1)+         ;PRINT CHAR
        DEC    R2              ;DECREASE COUNT
        BEQ    START          ;DONE IF COUNT = 0
        BR     OUTLOOP

BUFFER:

      .END    START
```

Rather than wait for the user to type something at INLOOP or wait for the output buffer to have available space at OUTLOOP, the routine can be recoded using .TTINR and .TTOUTR as follows:



# Programmed Requests

```

.MCALL ..V2,...,REGDEF,.TTYIN,.TTYOUT
..V2..
.REGDEF
.MCALL .TTINR,.TTOUTR,.EXIT

```

```

START:  MOV    #BUFFER,R1      ;POINT R1 TO BUFFER
        CLR    R2              ;CLEAR CHARACTER COUNT
        BIS    #100,0#44      ;WE REALLY WANT CARRY SET
INLOOP:  .TTINR                ;GET CHAR FROM TERMINAL
        BCS    NOCHAR         ;NONE AVAILABLE
        MOV    R0,(R1)+        ;PUT CHAR IN BUFFER
        INC    R2              ;INCREASE COUNT
        CMP    R0,#12         ;WAS LAST CHAR = LF?
        BNE    INLOOP         ;NO-GET NEXT
        MOV    #BUFFER,R1     ;YES-POINT R1 TO BUFFER
OUTLOOP: MOV    (R1),R0        ;PUT CHAR IN R0
        .TTOUTR               ;TYPE IT
        BCS    NOROOM         ;NO ROOM IN OUTPUT BUFFER
        DEC    R2              ;DECREASE COUNT
        BEQ    START          ;DONE IF COUNT=0
        INC    R1              ;BUMP BUFFER POINTER
        BR     OUTLOOP        ;AND TYPE NEXT
NOCHAR:

```

```

.TTINR                                ;PERIODIC CHECK FOR
BCC     CHRIN                        ;CHARACTER AVAILABILITY
;GOT ONE

```

```

.
.
(code to be executed
while waiting)
.
.

```

```

BR     NOCHAR

```

NOROOM:

```

MOV    (R1),R0                    ;PERIODIC ATTEMPT TO TYPE
.TTOUTR                                ;CHARACTER
BCC    CHROUT                      ;SUCCESSFUL

```

```

.REPT  10
NOP
.ENDR

```

```

TYPEIT: BIC    #100,0#44          ;THIS MUST GO OUT
;SO DON'T HANG WHILE
;WAITING FOR ROOM.
.TTYOUT (R1)
BIS    #100,0#44                  ;PUT CHAR
;RESTORE NO-WAIT

```

```

BUFFER: .BLKW  100.
.END     START

```



## Programmed Requests

### .TWAIT

#### 9.4.45 .TWAIT

The .TWAIT request suspends the user's job for an indicated length of time. .TWAIT requires a queue element, and thus, should be taken into account when the .QSET request is executed.

Macro call: .TWAIT .area, .time

where: .time is a pointer to two words of time  
(high-order first, low-order second).

#### Request Format

R0 → .area: 

|       |   |
|-------|---|
| 24    | 0 |
| .time |   |

#### Errors:

| <u>Code</u> | <u>Explanation</u>              |
|-------------|---------------------------------|
| 0           | No queue element was available. |

#### Example:

.TWAIT can be used in applications where a program must be only activated periodically. This example will 'wake up' every 10 seconds to perform a task, and then 'sleep' again.

```

.MCALL ..V2...REGDEF,.TWAIT,.QSET,.EXIT,.PRINT
..V2..
.REGDEF
GOI .QSET #QAREA,#7 ;SET UP 7 EXTRA ELEMENTS
START: MOV #EMTLST,R0 ;SET R0 TO THE ARG, BLOCK
.TWAIT ;GO TO SLEEP FOR 10 SECONDS
BCS NOQ ;NO QUEUE ELEMENT?
JSR PC,TASK ;DO SOMETHING HERE
BR START ;AND THEN CYCLE
QAREA: .BLKW 7*7 ;SPACE FOR 7 ELEMENTS
EMTLST: .BYTE 0,24
.WORD TIME
TIME: .WORD 0,10,*60, ;10 SECOND INTERVALS
TASK:

```



## Programmed Requests

```

INC      MPTR
BIT      #1,MPTR
BEQ      1$
,PRINT   #MSG
RTS      PC
1$: ,PRINT #MSG1
RTS      PC
MPTR:    ,WORD 0
MSG1:    ,ASCIZ /TICK/
MSG1:    ,ASCIZ /TOCK/
,EVEN
NOQ:     ,EXIT
,END     GO

```

.WAIT

### 9.4.46 .WAIT

The .WAIT request suspends program execution until all input/output requests on the specified channel are completed. The .WAIT request combined with the .READ/.WRITE requests makes double-buffering a simple process.

.WAIT also conveys information back through its error returns. An error is returned if either the channel is not currently open or if the last I/O operation resulted in a hardware error.

Macro Call: .WAIT .chan

Note:

In a F/B system, executing a .WAIT when I/O is pending causes that job to be suspended and the other job to run, if possible.

Request Format:

R0 ⇒ 0 .chan

Errors:

| <u>Code</u> | <u>Explanation</u>   |
|-------------|--|
| 0           | Channel specified is not open.   |
| 1           | Hardware error occurred on the previous I/O operation on this channel. |

These error codes make the .WAIT request useful in checking channel status.



## Programmed Requests

### Example:

For an example of .WAIT used for I/O synchronization, see the examples in the next section.

An example of the use of .WAIT for error detection is its use in conjunction with .CSIGEN to determine which file fields in the command string have been specified. For example, a program such as MACRO might use the following code to determine if a listing file is desired.

```
.MCALL ..V2,,,REGDEF,.WAIT,.CSIGEN,.EXIT
..V2,,
,REGDEF

START:

.CSIGEN #DSPACE,#DEXT,#0 /PROCESS COMMAND STRING
.WAIT #0 /CHECK FOR FILE IN FIRST FIELD
BCS NOBINARY /NO BINARY DESIRED

NOBINARY:

.WAIT #1 /CHECK FOR LISTING SPECIFICATION
BCS NOLISTING /NO LISTING DESIRED

NOLISTING:

.WAIT #3 /CHECK FOR INPUT FILE OPEN
BCS ERROR /NO INPUT FILE

ERROR: ,EXIT

DEXT: .RAD50 /MAC/
.RAD50 /OBJ/
.RAD50 /LST/
.WORD 0

DSPACE=.

.END START
```

**.WRITE/.WRITC/.WRITW**

#### 9.4.47 .WRITE/.WRITC/.WRITW

##### .WRITE

The .WRITE request transfers a specified number of words from memory to the specified channel. Control returns to the user program immediately after the request is queued.



## Programmed Requests

Macro Call: `.WRITE .area, .chan, .buff, .wcnt, .blk`

where: `.buff` is the address of the memory buffer to be used for output.

`.wcnt` is the number of words to be written.

`.blk` is the number of the block to be written.

### Request Format:

|             |       |       |
|-------------|-------|-------|
| R0 ⇒ .area: | 11    | .chan |
|             | .blk  |       |
|             | .buff |       |
|             | .wcnt |       |
|             | 1     |       |

### Notes:

See the note following `.WRITW`.

### Errors:

| <u>Code</u> | <u>Explanation</u>                   |
|-------------|--------------------------------------|
| 0           | Attempted to write past end-of-file. |
| 1           | Hardware error.                      |
| 2           | Channel was not opened.              |

### Example:

Refer to the examples following `.WRITW`.

### `.WRITC`

The `.WRITC` request transfers a specified number of words from memory to a specified channel. Control returns to the user program immediately after the request is queued. Execution of the user program continues until the `.WRITC` is complete, then control passes to the routine specified in the request. When an RTS PC is encountered in the routine, control returns to the user program.

Macro Call: `.WRITC .area, .chan, .buff, .wcnt, .crtn, .blk`

where: `.buff` is the address of the memory buffer to be used for output.

`.wcnt` is the number of words to be written.

`.crtn` is the address of the completion routine to be entered.

`.blk` is the number relative to the start of the file, not block 0 of the device. The monitor translates the block supplied into an absolute device block number. The user program normally updates `.blk` before it is used again.



## Programmed Requests

### Request Format:

|             |       |       |
|-------------|-------|-------|
| R0 ⇒ .area: | 11    | .chan |
|             | .blk  |       |
|             | .buff |       |
|             | .wcnt |       |
|             | .crtn |       |

When entering a .WRITC completion function the following are true:

1. R0 contains the channel status word for the operation. If bit 0 of R0 is set, a hardware error occurred during the transfer. The data may not be reliable.
2. R1 contains the octal channel number of the operation. This is useful when the same completion function is to be used for several different transfers.

### Notes:

See the note following .WRITW.

### Errors:

| <u>Code</u> | <u>Explanation</u>  |
|-------------|---|
| 0           | End-of-file on output. Tried to write outside limits of file. |
| 1           | Hardware error occurred.                                      |
| 2           | Specified channel is not open.                                |

### Example:

Refer to the examples following .WRITW.

### .WRITW

The .WRITW request transfers a specified number of words from memory to the specified channel. Control returns to the user program when the .WRITW is complete.

Macro Call: .WRITW .area, chan, .buff, .wcnt, .blk

|              |  |
|--------------|--|
| where: .buff | is the address of the buffer to be used for output.                |
| .wcnt        | is the number of words to be written. The number must be positive. |
| .blk         | is the number of the block to be written.                          |



## Programmed Requests

### Request Format:

|             |       |       |
|-------------|-------|-------|
| R0 → .area: | 11    | .chan |
|             | .blk  |       |
|             | .buff |       |
|             | .wcnt |       |
|             | 0     |       |

### Note:

Upon return from any of the .WRITE EMTs, R0 contains the number of words written. If an attempt is made to write past end-of-file but the request can be partially fulfilled, the output word count is shortened and an error is returned (except .READ, which gives an error only if no words can be transferred). Note that a write will be done, and a completion routine, if any, will be entered, unless the request cannot even be partially filled (shortened word count=0).

### Errors:

| Code | Explanation                  |
|------|------------------------------|
| 0    | Attempted to write past EOF. |
| 1    | Hardware error.              |
| 2    | Channel was not opened.      |

### Examples:

The following routine illustrates the differences between the three types of .READ/.WRITE requests and is coded in three ways, each using a different mode of monitor I/O. The routine itself is a simple program to duplicate a paper tape.

In the first example, .READW and .WRITW are used. The I/O is completely synchronous, with each request retaining control until the buffer is filled (or emptied).

```

.MCALL ..V2...REGDEF,.FETCH,.READW,.WRITW
.MCALL .ENTER,.LOOKUP,.PRINT,.EXIT,.CLOSE,.WAIT
..V2..
.REGDEF

```

ERRWD=52

```

START: .FETCH #HSPACE,#PRNAME ;GET PR HANDLER
      BCS FERR ;PR NOT AVAILABLE
      MOV R0,R2 ;R0 HAS NEXT FREE LOCATION
      .FETCH R2,#PPNAME ;GET PP HANDLER
      BCS FERR ;NOT AVAILABLE
      MOV #AREA,R5 ;EMT ARGUMENT AREA
      CLR R4 ;R4 IS OUTPUT CHANNEL; 0
      MOV #1,R3 ;R3 IS INPUT CHANNEL ;1
      .ENTER R5,R4,#PPNAME ;ENTER THE FILE
      BCS ENERR ;SOME ERROR IN ENTER
      .LOOKUP R5,R3,#PRNAME ;LOOKUP FILE ON CHANNEL ;1
      BCS LKERR ;ERROR IN LOOKUP
      CLR R1 ;USE R1 AS BLOCK NUMBER
LOOP: .READW R5,R3,#BUFF,#256,,R1 ;READ ONE BLOCK
      BCS RDERR
      .WRITW R5,R4,#BUFF,#256,,R1 ;WRITE THAT BLOCK

```



# Programmed Requests

```

BCS      WTERR
INC      R1

;BUMP BLOCK, NOTE: THIS IS
;NOT NECESSARY FOR NON-FILE
;DEVICES IN GENERAL, IT IS
;USED HERE AS AN EXAMPLE OF
;A GENERAL TECHNIQUE.
;KEEP GOING
;ERROR, IS IT EOF?
;YES
;NO, HARD READ ERROR

RDERR:   BR      LOOP
        TSTB    ERRWD
        BEQ     15
        .PRINT  #RMSG
        .EXIT
15:      .CLOSE  R3
        .CLOSE  R4
        .EXIT
        .PRINT  #WTMSG
        .EXIT
PRNAME:  .RAD50  /PR /
        .WORD   0
PPNAME:  .RAD50  /PP /
        .WORD   0
FERR:    .PRINT  #FMSG
        .EXIT
ENERR:   .PRINT  #EMSG
        .EXIT
LKERR:   .PRINT  #LMSG
        .EXIT
FMSG:    .ASCIZ  /NO DEVICE?/
EMSG:    .ASCIZ  /ENTRY ERROR?/
LMSG:    .ASCIZ  /LOOKUP ERROR?/
RMSG:    .ASCIZ  /READ ERROR?/
WTMSG:   .ASCIZ  /WRITE ERROR?/
        .EVEN
AREA:    .BLKW   10
BUFF:    .BLKW   256.
HSPACE=.
        .END    START

```

The same routine can be coded using .READ and .WRITE as follows. The .WAIT request is used to determine if the buffer is full or empty prior to its use.

```

.MCALL  ..V2...REGDEF,.FETCH,.READ,.WRITE
.MCALL  .ENTER,.LOOKUP,.PRINT,.EXIT,.CLOSE,.WAIT
..V2..
.REGDEF

ERRWD=52

START:  .FETCH  #HSPACE,#PRNAME ;GET PR HANDLER
BCS     FERR    ;PR NOT AVAILABLE
MOV     R0,R2   ;R0 HAS NEXT FREE LOCATION
.FETCH  R2,#PPNAME ;GET PP HANDLER
BCS     FERR    ;NOT AVAILABLE
MOV     #AREA,R5 ;EMT ARGUMENT AREA
CLR     R4      ;R4 IS OUTPUT CHANNEL, 0
MOV     #1,R3   ;R3 IS INPUT CHANNEL, 1
.ENTER  R5,R4,#PPNAME ;ENTER THE FILE

```



## Programmed Requests

```

      BCS      ENERR      ;SOME ERROR IN ENTER
      .LOOKUP  R5,R3,#PRNAME ;LOOKUP FILE ON CHANNEL 1
      BCS      LKERR      ;ERROR IN LOOKUP
      CLR      R1         ;USE R1 AS BLOCK NUMBER
LOOP:  .READ   R5,R3,#BUFF,#256,,R1 ;READ A BUFFER
      BCS      RDERR
      .WAIT   R3         ;WAIT FOR BUFFER
      BCS      IOERR      ;ERROR HERE IS HARD ERROR
      .WRITE  R5,R4,#BUFF,#256,,R1 ;WRITE THE BUFFER
      BCS      IOERR      ;I/O ERROR
      INC     R1
      BR      LOOP      ;KEEP GOING
RDERR: TSTB   ERRWD      ;ERROR, IS IT EOF?
      BNE     IOERR      ;NO, HARD ERROR
      .CLOSE  R3         ;CLOSE INPUT AND OUTPUT
      .CLOSE  R4
      .EXIT
IOERR: .PRINT  #IOMSG     ;AND EXIT,
      .EXIT          ;NO, HARD READ ERROR
PRNAME: .RAD50 /PR /      ;NOTE THAT PR NEEDS NO FILE NAME
      .WORD   0          ;FILE NAME NEED ONLY BE 0.
PPNAME: .RAD50 /PP /
      .WORD   0
FERR:  .PRINT  #FMSG      ;ERROR ACTIONS GO HERE, IT IS
      .EXIT          ;GENERALLY UNDESIRABLE TO
ENERR: .PRINT  #EMSG      ;EXECUTE A HALT OR RESET
      .EXIT          ;INSTRUCTION ON ERROR.
LKERR: .PRINT  #LMSG
      .EXIT
FMSG:  .ASCIZ  /NO DEVICE?/
EMSG:  .ASCIZ  /ENTRY ERROR?/
LMSG:  .ASCIZ  /LOOKUP ERROR?/
IOMSG: .ASCIZ  "I/O ERROR?"
WTMSG: .ASCIZ  /WRITE ERROR?/
      .EVEN
AREA:  .BLKW   10
BUFF:  .BLKW   256.
HSPACE*.
      .END      START

```

.READ and .WRITE are also often used for double-buffered I/O. The basic double-buffering algorithm for input is:

### Explanation

|       |      |          |                                       |
|-------|------|----------|---------------------------------------|
| LOOP: | READ | BUFFER 1 | Fill BUFFER 1                         |
|       | WAIT | BUFFER 1 | Wait for BUFFER 1 to fill             |
|       | READ | BUFFER 2 | Start filling BUFFER 2                |
|       | USE  | BUFFER 1 | Process BUFFER 1 while BUFFER 2 fills |
|       | WAIT | BUFFER 2 | Wait for BUFFER 2 to fill             |
|       | READ | BUFFER 1 | Start filling BUFFER 1                |
|       | USE  | BUFFER 2 | Process BUFFER 2 while BUFFER 1 fills |
|       | BR   | LOOP     |                                       |



## Programmed Requests

Correspondingly, the basic double-buffering algorithm for output is:

|       |                | <u>Explanation</u>                   |
|-------|----------------|--------------------------------------|
| LOOP: | FILL BUFFER 1  | Prepare BUFFER 1 for output          |
|       | WRITE BUFFER 1 | Start emptying BUFFER 1              |
|       | FILL BUFFER 2  | Fill BUFFER 2 while BUFFER 1 empties |
|       | WAIT BUFFER 1  | Wait for BUFFER 1 to empty           |
|       | WRITE BUFFER 2 | Start emptying BUFFER 2              |
|       | FILL BUFFER 1  | Fill BUFFER 1 while BUFFER 2 empties |
|       | WAIT BUFFER 2  | Wait for BUFFER 2 to empty           |
|       | BR LOOP        |                                      |

This example program can be coded using completion routines via .READC and .WRITC as follows. Once the initial read is performed, the remainder of the I/O is performed by the completion routines.

```
.MCALL ..V2...REGDEF,,FETCH,,READC,,WRITC
.MCALL .ENTER,,LOOKUP,,PRINT,,EXIT,,CLOSE,,WAIT
..V2..
.REGDEF
```

ERRWD=52

```
START:  ,FETCH  #MSPACE,#PRNAME  ;GET PR HANDLER
        BCS    FERR              ;PR NOT AVAILABLE
        MOV    R0,R2             ;R0 HAS NEXT FREE LOCATION
        ,FETCH  R2,#PPNAME       ;GET PP HANDLER
        BCS    FERR              ;NOT AVAILABLE
        MOV    #AREA,R5         ;EMT ARGUMENT AREA
        CLR    R4                ;R4 IS OUTPUT CHANNEL; 0
        MOV    #1,R3             ;R3 IS INPUT CHANNEL ;1
        ,ENTER  R5,R4,#PPNAME    ;ENTER THE FILE
        BCS    ENERR             ;SOME ERROR IN ENTER
        ,LOOKUP R5,R3,#PRNAME    ;LOOKUP FILE ON CHANNEL 1
        BCS    LKERR             ;ERROR IN LOOKUP
        CLR    R1                ;USE R1 AS BLOCK NUMBER
LOOP:   CLR    DFLG              ;CLEAR DONE/ERROR FLAG
        ,READC  R5,R3,#BUFF,#256,,#RDCOMP,R1 ;READ ONE BLOCK
        BCS    EOF              ;NO ERROR WILL HAPPEN HERE
1$      TST    DFLG              ;DONE FLAG SET?
        BEQ    1$               ;NO, WAIT FOR IT TO BE SET.
        BMI    IDERR            ;YES, BUT HARD ERROR OCCURRED
EOF:    ,CLOSE  R3               ;CLOSE INPUT AND OUTPUT CHANNELS
        ,CLOSE  R4
        ,EXIT
,ENABL  LSB
RDCOMP: ROR    R0                ;IF BIT 0 SET
        BCS    RDERR            ;AN ERROR OCCURRED.
        ,WRITC  R5,R4,#BUFF,#256,,#WRCOMP,R1 ;WRITE THAT BLOCK
        BCC    2$               ;ERROR HERE IS HARDWARE
RDERR:  MOV    #-1,DFLG          ;FLAG THE ERROR
2$      RTS    PC
WRCOMP: ROR    R0
        BCS    RDERR            ;HARDWARE ERROR
        INC    R1               ;BUMP BLOCK.
        ,READC  R5,R3,#BUFF,#256,,#RDCOMP,R1
        BCC    3$               ;NO ERROR
        TSTB   ERRWD            ;EOF?
```



# Programmed Requests

```

35:      BNE      RDERR      ;NO, HARD ERROR
        INC      DFLG      ;SAY WE'RE DONE
        RTS      PC

.DSABL   LSB
DPLG1:   .WORD    0
PRNAME:  .RAD50   /PR /
        .WORD    0
PPNAME:  .RAD50   /PP /
        .WORD    0
IOERR:   .PRINT   #IOMSG
        .EXIT
FERR:    .PRINT   #FMSG      ;ERROR ACTIONS GO HERE. IT IS
        .EXIT              ;GENERALLY UNDESIRABLE TO
ENERR:   .PRINT   #EMSG      ;EXECUTE A HALT OR RESET
        .EXIT              ;INSTRUCTION ON ERROR.
LKERR:   .PRINT   #LMSG
        .EXIT
FMSG:    .ASCIZ   /NO DEVICE?/
EMSG:    .ASCIZ   /ENTRY ERROR?/
LMSG:    .ASCIZ   /LOOKUP ERROR?/
IOMSG:   .ASCIZ   "I/O ERROR?"
        .EVEN
AREA:    .BLKW    10
BUFF:    .BLKW    256.
HSPACE:  .
        .END      START

```

The following example incorporates the .LOOKUP, .READW, and .CLOSE requests. The program opens the file RT11.MAC which is on the system device, SY:, for input on channel 0. The first block is read and the file is then closed.

```

.MCALL   ..V2...REGDEF,,CLOSE,,LOOKUP
.MCALL   .PRINT,,EXIT,,READW,,FETCH

..V2..
.REGDEF

START:   MOV      #LIST,R5      ;EMT ARGUMENT LIST POINTER
        CLR      R4            ;BLOCK NUMBER
        CLR      R3            ;CHANNEL #
        .FETCH   #CORADD,#FPTR ;FETCH DEVICE HANDLER
        BCC      25
        MOV      #FETMSG,R0     ;FETCH ERROR
15:      .PRINT   ;PRINT ERROR MESSAGE
        .EXIT
25:      .LOOKUP  R5,R3,#FPTR    ;LOOKUP FILE ON CHANNEL 0
        BCC      35
        MOV      #LKMSG,R0     ;PRINT FAILURE MESSAGE
        BR       15
35:      .READW   R5,R3,#BUFF,#256,,R4 ;REA ONE BLOCK
        BCC      45
        MOV      #HOMSG,R0     ;READ ERROR
        BR       15
45:      .CLOSE   R3            ;CLOSE THE CHANNEL
        .EXIT

LIST:    .BLKW    5              ;LIST FOR EMT CALLS
FPTR:    .RAD50   /SY RT11 MAC/ ;RAD50 OF FIEL NAME,DEVICE
FETMSG:  .ASCIZ   /FETCH FAILED/ ;ASCII MESSAGES
LKMSG:   .ASCIZ   /LOOKUP FAILED/

```



## Programmed Requests

```
RDMSG: .ASCIZ /READ FAILED/
        .EVEN
CORADD: .BLKW  2000          /SPACE FOR LARGEST HANDLERS
BUFF: .
        .END  START
```

### 9.5 CONVERTING VERSION 1 MACRO CALLS TO VERSION 2

As mentioned in the introduction of this chapter, RT-11 Version 2 supports a slightly modified format for system macro calls than Version 1. This section details the conversion process from the Version 1 format to Version 2.

#### 9.5.1 Macro Calls Requiring No Conversion

Version 1 macro calls which need no conversion are:

|          |          |
|----------|----------|
| .CSIGN   | .RCTLO   |
| .CSISPC  | .RELEAS  |
| .DATE    | .SETTOP* |
| .DSTATUS | .SRESET  |
| .EXIT    | .TTINR** |
| .FETCH   | .TTOUR   |
| .HRESET  | .TTYIN   |
| .LOCK    | .TTYOUT  |
| .PRINT   | .UNLOCK  |
| .QSET    |          |

\*Provided location 50 is examined for the maximum value.

\*\*Except in F/B System.

#### 9.5.2 Macro Calls Which May Be Converted

The following Version 1 macro calls may be converted:

|         |            |
|---------|------------|
| .CLOSE  | .RENAME    |
| .DELETE | .REOPEN    |
| .ENTER  | .SAVSTATUS |
| .LOOKUP | .WAIT      |
| .READ   | .WRITE     |

The general format of the V1 macro is:

```
.PRGREQ .chan, .arg1 .arg2,...argn
```

In this form, .chan is an integer between 0 and 17 (inclusive), and is not a general assembler argument. The channel number is assembled into the EMT instruction itself. The arguments arg1-argn are always pushed either into R0 or on the stack.

The V2 equivalent of the above call is:

```
.PRGREQ .area, .chan, .arg1,...argn
```



## Programmed Requests

In the V2 call, the .chan argument can be any legal assembler argument; it need not be in the range 0 to 17 (octal), but should be in the range 0-377 (octal). .area points to a memory list where the arguments arg1...argn will go.

As an example, consider a .READ request in both forms:

```
V1:      .READ 5,#BUFF,#256.,BLOCK
V2:      .READ #AREA,#5,#BUFF,#256.,BLOCK
          .
          .
          .
AREA:     .WORD 0    ;CHANNEL/FUNCTION CODE HERE
          .WORD 0    ;BLOCK NUMBER HERE
          .WORD 0    ;BUFFER ADDRESS HERE
          .WORD 0    ;WORD COUNT HERE
          .WORD 0    ;A 1 GOES HERE.
```

Thus, the difference in the calls is that in Version 2 the channel number becomes a legal assembler argument and the .area argument has been added.

Following is a complete list of the conversions necessary for each of the EMT calls. Both the Version 1 and Version 2 formats are given. Note that parameters inside [] are optional parameters. Refer to the appropriate section in this chapter for more details of each request.

| <u>Version</u> | <u>Programmed Request</u>                   |
|----------------|---|
| V1:            | .DELETE .chan,.dblk                         |
| V2:            | .DELETE .area,.chan,.dblk,[.count]          |
| V1:            | .LOOKUP .chan,.dblk                         |
| V2:            | .LOOKUP .area,.chan,.dblk,[.count]          |
| V1:            | .ENTER .chan,.dblk,[.length]                |
| V2:            | .ENTER .area,.chan,.dblk,[.length],[.count] |
| V1:            | .RENAME .chan,.dblk                         |
| V2:            | .RENAME .area,.chan,.dblk                   |
| V1:            | .SAVESTAT .chan,.cblk                       |
| V2:            | .SAVESTAT .area,.chan,.cblk                 |
| V1:            | .REOPEN .chan,.cblk                         |
| V2:            | .REOPEN .area,.chan,.cblk                   |
| V1:            | .CLOSE .chan                                |
| V2:            | .CLOSE .chan                                |
| V1:            | .READ/.READW .chan,.buff,.wcnt,.blk         |
| V2:            | .READ/.READW .area,.chan,.buff,.wcnt,.blk   |
| V1:            | .READC .chan,.buff,.wcnt,.crtn,.blk         |
| V2:            | .READC .area,.buff,.wcnt,.crtn,.blk         |
| V1:            | .WRITE/.WRITW .chan,.buff,.wcnt,.blk        |
| V2:            | .WRITE/.WRITW .area,.chan,.buff,.wcnt,.blk  |



## Programmed Requests

V1: .WRITC .chan,.buff,.wcnt,.crtn,.blk  
V2: .WRITC .area,.chan,.buff,.wcnt,.crtn,.blk

V1: .WAIT .chan  
V2: .WAIT .chan

Important features to keep in mind for Version 2 calls are:

1. Version 2 calls require the .area argument, which points to the area where the other arguments will be.
2. Enough memory space must be allocated to hold all the required arguments.
3. .chan must be any legal assembler argument, not just an integer between 0-17 (octal).
4. Blank fields are permitted in the Version 2 calls. Any field not specified (left blank) is left alone in the argument block.



## CHAPTER 10

### EXPAND UTILITY PROGRAM

EXPAND is an RT-11 system program which processes the macro references in a macro assembly language source file. EXPAND accepts a subset of the complete macro language and, using the system library file SYSMAC.8K, produces an output file in which all legal macro references are expanded into macro-free source code. EXPAND is normally used with ASEMBL, the RT-11 assembler designed for minimum memory configurations (refer to Chapter 11).

#### 10.1 LANGUAGE

EXPAND simply copies its input files to its output file unless it encounters any of the following directives (see Chapter 5 for more information about these directives):

1. .MCALL Directs EXPAND to search the file SY:SYSMAC.SML to find the macro names listed in the .MCALL directive. If the macro names are found, EXPAND stores their definitions in its internal tables.
2. .MACRO Directs EXPAND to copy a macro definition from the user's input file into the internal tables.
3. .name If .name is the name of a macro defined in either a .MCALL or .MACRO directive, then .name is expanded according to the definition stored for it in the EXPAND internal tables.
4. .ENDM If encountered while storing a macro definition, the .ENDM directive terminates the definition. It is not recognized outside macro definitions.

#### 10.2 RESTRICTIONS

Unlike the full macro assembler (MACRO), EXPAND only expands macros that observe the following restrictions:



## EXPAND Utility Program

1. The following directives may not be used:

|         |        |
|---------|--------|
| .ERROR  | .NARG  |
| .IF DIF | .NCHR  |
| .IF IDN | .NTYPE |
| .IRP    | .PRINT |
| .IRPC   | .REPT  |
| .MEXIT  |        |

2. Macros cannot be nested. Recursive macros that call themselves directly or indirectly are illegal and cause an error message.
3. Macros cannot be redefined. Once a name has been used for a macro name, it cannot be used again in the program for a macro or symbol name.
4. Macro names must begin with a dot (.). If the dot is missing, an error message is printed.
5. Dummy argument names must begin with a dot (.). Such names cannot be used as dummy argument names in the macro but can be used for other purposes outside of the macro.
6. The backslash operator is not available.
7. Automatically created symbols are not available.
8. No more than 30 arguments may be used in any MACRO directive.

### 10.3 CALLING AND USING EXPAND

To run EXPAND, type:

R EXPAND

in response to the dot printed by the Keyboard Monitor. EXPAND responds with an asterisk indicating that it is ready to accept a command string. A command string must be of the following form:

\*ofile=ifile1,ifile2,...,ifile6

ifile2 through ifile6 are optional. Each file specification follows the general RT-11 command string syntax (dev:filnam.ext). The default value for each file specification is noted below:

| <u>I/O File</u>       | <u>Dev</u>  | <u>Ext</u> |
|-----------------------|---|------------|
| ofile                 | DK  | PAL        |
| ifile1,...,<br>ifile6 | device used<br>for last source<br>file specified<br>or DK | MAC        |

Type CTRL C to halt EXPAND and return control to the monitor. To restart EXPAND, type R EXPAND or the REENTER command in response to the monitor's dot.



## EXPAND Utility Program

EXPAND copies sequentially the specified input files to the specified output file until a macro directive is encountered. EXPAND then changes the macro directive to a comment by inserting a semicolon so that it will not be seen later by the assembler (usually ASEMBL).

If the directive is .MCALL, EXPAND searches the system library file (SYSMAC.8K) for the requested macro definitions. The requested definitions are then included in the user's program in the order in which they are found in the library.

For the .MACRO directive, EXPAND reads each line following the directive up to the next .ENDM directive. Each line is stored in the internal definition table and then changed to a comment in the output file so that it is not processed later by the assembler. Also, any occurrence of a macro argument name within the definition is flagged internally so that it can be replaced by the real argument value whenever the macro is later referenced.

For macro references, EXPAND locates the stored macro definition in its internal tables, binds the actual argument values to the argument names, and changes the macro reference to a comment line. EXPAND then begins copying the stored definition to the output file. Whenever a macro argument name is encountered in the definition, it is replaced by the corresponding actual argument value.

### Examples:

The following are examples of input and corresponding EXPAND output.

| <u>INPUT</u>         | <u>OUTPUT</u>                            |
|----------------------|--|
| R1=X1                | R1=X1                                    |
| SP=X6                | SP=X6                                    |
| PC=X7                | PC=X7                                    |
| .MACRO .CALL .SUBR   | , .MACRO .CALL .SUBR                     |
| JSR PC,.SUBR         | , JSR PC,.SUBR                           |
| .ENDM                | , .ENDM                                  |
| .MCALL .LOOKUP,.READ | , .MCALL .LOOKUP,.READ                   |
|                      | ,.MACRO .LOOKUP .AREA,.CHAN,.DEVBLK,.SPF |
|                      | ,.IF NDF ...V2                           |
|                      | ,.IF NB .CHAN                            |
|                      | , MOV .CHAN,%0                           |
|                      | ,.ENDC                                   |
|                      | , EMT A0<20+.AREA>                       |
|                      | ,.IFF                                    |
|                      | ,.IF NB .AREA                            |
|                      | , MOV .AREA,%0                           |
|                      | , MOV #400,(0)                           |
|                      | ,.ENDC                                   |
|                      | ,.IF NB .CHAN                            |
|                      | , MOVB .CHAN,(0)                         |
|                      | ,.ENDC                                   |
|                      | ,.IF NB .DEVBLK                          |
|                      | , MOV .DEVBLK,2,(0)                      |
|                      | ,.ENDC                                   |
|                      | ,.IF NB .SPF                             |
|                      | , MOV .SPF,4,(0)                         |
|                      | ,.IFF                                    |
|                      | , CLR 4,(0)                              |
|                      | ,.ENDC                                   |



# EXPAND Utility Program

```

                                EMT      ^0375
                                / .ENDC
                                / .ENDM
                                / .MACRO .READ      .AREA,.CHAN,.BUFF,.WCNT,.BLK
                                / .IF NOF ...V2
                                / .IF NB .WCNT
                                /
                                / .ENDC
                                /
                                /      MOV      .WCNT,%0
                                /
                                /      MOV      #1,-(6.)
                                /      MOV      .BUFF,-(6.)
                                /      MOV      .CHAN,-(6.)
                                /      EMT      ^0<200+.AREA>
                                / .IFF
                                / .IF NB .AREA
                                /
                                /      MOV      .AREA,%0
                                /      MOV      #4000,(0)
                                / .ENDC
                                / .IF NB .CHAN
                                /
                                /      MOV      .CHAN,(0)
                                / .ENDC
                                / .IF NB .BLK
                                /
                                /      MOV      .BLK,2.(0)
                                / .ENDC
                                / .IF NB .BUFF
                                /
                                /      MOV      .BUFF,4.(0)
                                / .ENDC
                                / .IF NB .WCNT
                                /
                                /      MOV      .WCNT,6.(0)
                                / .ENDC
                                /
                                /      MOV      #1,8.(0)
                                /      EMT      ^0375
                                / .ENDC
                                / .ENDM

                                .CSECT MAIN
                                .GLOBL SQRT
STACK:  .BLKW 100
BUFR:   .BLKW 100
INBLK:  .BLKW 10
START:  MOV #STACK,SP
A:      MOV R1,-(SP)
B:      .CALL SQRT
        .LOOKUP 0,#INBLK

                                .CSECT MAIN
                                .GLOBL SQRT
STACK:  .BLKW 100
BUFR:   .BLKW 100
INBLK:  .BLKW 10
START:  MOV #STACK,SP
A:      MOV R1,-(SP)
B:      .CALL SQRT
        JSR PC,SQRT
        .LOOKUP 0,#INBLK
                                /
                                / .IF NOF ...V2
                                / .IF NB #INBLK

```



# EXPAND Utility Program

```

                                MOV      #INBLK,X0
                                EMT      ^0<20+0>
                                .IFF
                                .IF NB 0
                                MOV      0,X0
                                MOV      #400,(0)
                                .ENDC
                                .IF NB #INBLK
                                MOV      #INBLK,(0)
                                .ENDC
                                .IF NB
                                MOV      ,2,(0)
                                .ENDC
                                .IF NB
                                MOV      ,4,(0)
                                .IFF
                                CLR      4,(0)
                                .ENDC
                                EMT      ^0375
                                .ENDC
                                CLR R1 ,BLOCK NUMBER
                                .READ 0,#BUFR,#256.,R1
                                CLR R1 ,BLOCK NUMBER
                                .READ 0,#BUFR,#256.,R1
                                .IF NDF ...V2
                                .IF NB R1
                                MOV      R1,X0
                                .ENDC
                                MOV      #1,-(6.)
                                MOV      #256.,-(6.)
                                MOV      #BUFR,-(6.)
                                EMT      ^0<200+0>
                                .IFF
                                .IF NB 0
                                MOV      0,X0
                                MOV      #4000,(0)
                                .ENDC
                                .IF NB #BUFR
                                MOV      #BUFR,(0)
                                .ENDC
                                .IF NB
                                MOV      ,2,(0)
                                .ENDC
                                .IF NB #256.
                                MOV      #256.,4.(0)
                                .ENDC
                                .IF NB R1
                                MOV      R1,6.(0)
                                .ENDC
                                MOV      #1,8.(0)
                                EMT      ^0375
                                .ENDC
                                HALT
                                .END START
                                HALT
                                .END START

```



## EXPAND Utility Program

### 10.4 EXPAND ERROR MESSAGES

The following messages are caused by fatal errors detected by EXPAND. They print on the console terminal and cause EXPAND to restart:

| <u>Message</u>                 | <u>Explanation</u>  |
|--------------------------------|---|
| ?BAD SWITCH?                   | An unrecognized command string switch was specified.                                      |
| ?INPUT ERROR?                  | Hardware error in reading an input file.  |
| ?INSUFFICIENT CORE?            | Not enough memory to store macro definitions.   |
| ?MISSING END IN MACRO?         | End of input was encountered while storing a macro definition; probably missing an .ENDM. |
| ?NO INPUT FILE?                | There must be at least one input file.  |
| ?OUTPUT DEVICE FULL?           | No room to continue writing output; try to compress the device with PIP.                  |
| ?WRONG NUMBER OF OUTPUT FILES? | There must be exactly one output file.  |

The following errors are non-fatal but indicate that something is wrong in the input file(s). These errors appear in the output file as a line in the following form:

?\*\*\* ERROR \*\*\* message

After each run of EXPAND, the total number of non-fatal errors is printed on the console terminal.

| <u>Message</u>             | <u>Explanation</u>   |
|----------------------------|--|
| BAD MACRO ARG              | The macro argument is not formatted correctly.   |
| LINE TOO LONG              | A line has become longer than 132 characters.  |
| MACRO ALREADY DEFINED      | A macro was defined more than once.  |
| MACRO(S) NOT FOUND         | Macros listed in an .MCALL statement were not found in SYSMAC.8K (make sure SYSMAC.8K is present on system).   |
| MISSING COMMA IN MACRO ARG | Found spaces or tabs within a macro argument; try using brackets around the argument, e.g., <arg with spaces>. |



## EXPAND Utility Program

### MISSING DOT

A macro name or argument name does not begin with a dot.

### NAME DOESN'T MATCH

Optional name given in .ENDM directive does not match name given in corresponding .MACRO directive.

### NESTED MACROS

A macro is being defined or invoked within another macro.

### NO NAME

A macro definition has no name.

### SYNTAX

A macro directive is not constructed correctly.

### TOO MANY ARGS

A macro directive has more than 30 arguments.



1. The first part of the report is a general introduction to the subject of the study.

2. The second part of the report is a detailed description of the methods used in the study.

3. The third part of the report is a presentation of the results of the study.

4. The fourth part of the report is a discussion of the results and their implications.

5. The fifth part of the report is a conclusion and a list of references.



## CHAPTER 11

### ASEMBL, THE 8K ASSEMBLER

ASEMBL is designed for use on an RT-11 system with minimum memory space (or larger systems where system table space is critical) and is a subset of the RT-11 MACRO assembler described in Chapter 5. ASEMBL has the same features as MACRO with the following exceptions:

1. MACRO directives (.MACRO, .MCALL, .ENDM, .IRP, etc.) are not recognized
2. DATE is not printed in listings
3. Wide line-printer output is not available
4. There is no lower-case mode
5. There is no enable/disable punch directive
6. There are no floating point directives
7. There are no local symbols or local symbol blocks
8. CREF is not available

Many of the macro features are supported by the EXPAND program (as described in Chapter 10).

#### 11.1 CALLING AND USING ASEMBL

ASEMBL is loaded in response to the dot printed by the Keyboard Monitor with the RT-11 monitor R Command as follows:

R ASEMBL

followed by the RETURN key. ASEMBL responds with an asterisk (\*) and waits for specification of the output and input files in the standard RT-11 format as follows:

\*object,listing=source1,...,source6



## ASEMBL, the 8K Assembler

where:

object is a binary object file output by ASEMBL.

listing is the assembly listing file containing the assembly listing and symbol table.

source1,...,source6 are the ASCII source files containing the ASEMBL source program(s). A maximum of six source files is allowed.

A null specification in any of the file fields signifies that the associated input or output file is not desired. ASEMBL file specifications follow the standard RT-11 convention (dev:filnam.ext). The default value for each file specification is noted below:

| <u>I/O File</u>      | <u>Dev</u>                                       | <u>Ext</u> |
|----------------------|--|------------|
| object               | DK   | .OBJ       |
| listing              | device used for object output                    | .LST       |
| source1,..., source6 | device used for last source file specified or DK | .PAL       |

Type CTRL C to halt ASEMBL and return control to the monitor. To restart ASEMBL, type R ASEMBL or the REENTER command in response to the monitor's dot.

Table 11-1 lists the RT-11 macro directives which are not available in ASEMBL.

Table 11-1  
Directives not Available in ASEMBL

| Directive                           | Explanation  |
|-------------------------------------|--|
| .MACRO<br>.ENDM<br>.MEXIT<br>.MCALL | Macros cannot be defined in ASEMBL.                                      |
| .NCHR                               | The number of characters in an argument cannot be obtained with a macro. |
| .NARG                               | The number of arguments in a macro cannot be obtained with a macro.      |
| \ (backslash)                       | Symbols used as macro arguments cannot be passed as a numeric string.    |

(continued on next page)



Table 11-1 (Cont.)  
Directives not Available in ASEMBL

| Directive  | Explanation   |
|--|---|
| .ERROR   | Messages cannot be flagged with a P error code output as part of the assembly listing. Comment lines can be used to replace .ERROR. |
| .IF IDN<br>.IF DIF }   | Strings cannot be compared.   |
| .IRP<br>.IRPC }  | Indefinite repeat blocks cannot be created.   |
| .NTYPE   | A macro cannot be modified based on the addressing mode of an argument.   |
| .PRINT   | Messages cannot be output as part of the assembly listing. Comment lines can be used to replace .PRINT.                             |
| .REPT  | A block of code cannot be duplicated a number of times in-line with other source code using a directive.                            |
| .LIST ME<br>.NLIST ME<br>.LIST MEB<br>.NLIST MEB<br>.LIST MD<br>.NLIST MD<br>.LIST MC<br>.NLIST MC } | These directives have no effect.  |
| .LIST TTM<br>.NLIST TTM }  | Terminal mode is standard and cannot be changed.  |
| .ENABL LC<br>.DSABL LC }   | All lower case ASCII input is converted to upper case.  |
| .ENABL LSB<br>.DSABL LSB }   | Local symbols and local symbol blocks are not available in ASEMBL.  |
| .ENABLE PNC<br>.DSABL PNC }  | Binary output is always enabled.  |
| .ENABL FPT<br>.DSABL FPT<br>.FLT2<br>.FLT4 }   | Floating point directives are not available.  |

Example:

This example uses the output produced by the EXPAND program as input to ASEMBL. The assembly listing follows.



# ASEMBL, the 8K Assembler

```

1          ; RT=11 MACRO EXPAND V02=02
2
3
4          000001      R1=X1
5          000006      SP=X6
6          000007      PC=X7
7          ;          .MACRO .CALL .SUBR
8          ;          JSR PC,.SUBR
9          ;          .ENDM
10         ;          .MCALL .LOOKUP,.READ
11         ;          .MACRO .LOOKUP .AREA,.CHAN,.DEVBLK,.SPF
12         ;          .IF NDF ...V2
13         ;          .IF NB .CHAN
14         ;          MOV      .CHAN,X0
15         ;          .ENDC
16         ;          EMT      ^0<20+.AREA>
17         ;          .IFF
18         ;          .IF NB .AREA
19         ;          MOV      .AREA,X0
20         ;          MOV      #400,(0)
21         ;          .ENDC
22         ;          .IF NB .CHAN
23         ;          MOV      .CHAN,(0)
24         ;          .ENDC
25         ;          .IF NB .DEVBLK
26         ;          MOV      .DEVBLK,2.(0)
27         ;          .ENDC
28         ;          .IF NB .SPF
29         ;          MOV      .SPF,4.(0)
30         ;          .IFF
31         ;          CLR      4.(0)
32         ;          .ENDC
33         ;          EMT      ^0375
34         ;          .ENDC
35         ;          .ENDM
36         ;          .MACRO .READ .AREA,.CHAN,.BUFF,.WCNT,.BLK
37         ;          .IF NDF ...V2
38         ;          .IF NB .WCNT
39         ;          MOV      .WCNT,X0
40         ;          .ENDC
41         ;          MOV      #1,-(6.)
42         ;          MOV      .BUFF,-(6.)
43         ;          MOV      .CHAN,-(6.)
44         ;          EMT      ^0<200+.AREA>
45         ;          .IFF
46         ;          .IF NB .AREA
47         ;          MOV      .AREA,X0
48         ;          MOV      #4000,(0)
49         ;          .ENDC
50         ;          .IF NB .CHAN
51         ;          MOV      .CHAN,(0)
52         ;          .ENDC
53         ;          .IF NB .BLK
54         ;          MOV      .BLK,2.(0)
55         ;          .ENDC
56         ;          .IF NB .BUFF
57         ;          MOV      .BUFF,4.(0)

```



ASEMBL, the 8K Assembler

.MAIN. RT=11 MACRO VS02=00 PAGE 1+

```

58          J.ENDC
59          J.IF NB .WCNT
60          J          MOV      .WCNT,6.(0)
61          J.ENDC
62          J          MOV      #1,8.(0)
63          J          EMT      A0375
64          J.ENDC
65          J.ENDM
66          000000'      .CSECT MAIN
67                      .GLOBL SQRT
68 000000      STACK:   .BLKW 100
69 002000      BUFR:    .BLKW 100
70 004000      INBLK:   .BLKW 10
71 00420 012706 START:  MOV #STACK,8P
                      000000'
72 00424 010146 A1     MOV R1,-(8P)
73 00426      B11      .CALL SQRT
74 00426 004767      JSR PC,SQRT
                      000000G

75          J          .LOOKUP 0,#INBLK
76          .IF NDF ...V2
77          .IF NB #INBLK
78 00432 012700      MOV      #INBLK,X0
                      000400'
79          .ENDC
80 00436 104020      EMT      A0<20+0>
81          .IFF
82          .IF NB 0
83          MOV      0,X0
84          MOV      #400,(0)
85          .ENDC
86          .IF NB #INBLK
87          MOV8     #INBLK,(0)
88          .ENDC
89          .IF NB
90          MOV      ,2.(0)
91          .ENDC
92          .IF NB
93          MOV      ,4.(0)
94          .IFF
95          CLR      4.(0)
96          .ENDC
97          EMT      A0375
98          .ENDC
99 00440 005001      CLR R1 /BLOCK NUMBER
100          J          .READ 0,#BUFR,#256.,R1
101          .IF NDF ...V2
102          .IF NB R1
103 00442 010100      MOV      R1,X0
104          .ENDC
105 00444 012746      MOV      #1,-(6.)
                      000001
106 00450 012746      MOV      #256.,-(6.)
                      000400
107 00454 012746      MOV      #BUFR,-(6.)
                      000200
108 00460 104200      EMT      A0<200+0>

```



ASEMBL, the 8K Assembler.

,MAIN. RT=11 MACRU VS02=08 PAGE 1+

```

109      .IFF
110      .IF NB 0
111          MOV      0,X0
112          MOV      #4000,(0)
113      .ENDC
114      .IF NB #BUFR
115          MOV      #BUFR,(0)
116      .ENDC
117      .IF NB
118          MOV      ,2.(0)
119      .ENDC
120      .IF NB #256.
121          MOV      #256.,4.(0)
122      .ENDC
123      .IF NB R1
124          MOV      R1,6.(0)
125      .ENDC
126          MOV      #1,8.(0)
127          EMT      ^0375
128      .ENDC
129 0462 000000      HALT
130      000420'      .END START

```

,MAIN. RT=11 MACRU VS02=08 PAGE 1+  
SYMBOL TABLE

```

A      000424R      002  B      000426R      002  BUFR      000200R      002
INBLK  000400R      002  PC      =X000007      R1      =X000001
SP      =X000006      SQRT  = ***** G      STACK  000000R      002
START  000420R      002
. ABS.  000000      000
        000000      001
MAIN    000464      002
ERRORS DETECTED: 0
FREE CORE: 19117. WORDS

,LP:CH11.PAL

```

## 11.2 ASEMBL ERROR MESSAGES

The system error messages output for ASEMBL are abbreviated as follows:

| <u>Abbreviation</u> | <u>Explanation</u>   |
|---------------------|--|
| ?BSW?               | The switch specified was not recognized by the program.  |
| ?CORE?              | There are too many symbols in the program being assembled. Try dividing program into separately assembled subprograms. |
| ?NIF?               | No input file was specified and there must be at least one input file.   |
| ?ODF?               | No room to continue writing output; try to compress device with PIP.   |
| ?TMO?               | Too many output files were specified.  |



## APPENDIX A

### ASSEMBLY, LINK, AND BUILD INSTRUCTIONS

This appendix describes:

1. The system building procedure from each of the possible distribution media
2. Customization of the system for special hardware (LA30's, VT05B, 7-Track Magtape, varying numbers of RFl1 platters, 50-cycle clock rates)
3. Optimizations which can be affected at system build time
4. Assembly and linking instructions for system components

#### NOTE

This appendix assumes the user has read and is familiar with the entire RT-11 SYSTEM REFERENCE MANUAL, especially Chapter 2 (System Communication), Chapter 4 (PIP), and Chapter 6 (LINK). For step-by-step instructions for building the system the first time, refer to GETTING STARTED WITH RT-11 (DEC-11-ORCPA-D-D).

#### A.1 BUILDING RT-11 SYSTEMS

RT-11 is designed so that the monitor and device handlers which comprise the system are files on the system device. These files (called system files) all have the extension ".SYS", and can be manipulated between devices just like any other RT-11 file.

The running version of the monitor must be named MONITR.SYS; other versions of the monitor may reside on the system device, but they must be named something other than MONITR.SYS. The handlers for the system must be named xx.SYS, where xx is the device mnemonic as used in command strings. For example, the high-speed reader handler must be named PR.SYS. There may be many versions of a given handler on the system device, but the one that is in use must be named as above.



## Assembly, Link, and Build Instructions

Once copies of the system files have been obtained, the procedure for building an RT-11 system consists of the following basic steps:

1. Initializing the target device with an RT-11 directory
2. Transferring the appropriate monitor file to the target device and giving it the name MONITR.SYS
3. Transferring the appropriate handler files to the target device
4. Writing the appropriate bootstrap on the target device
5. Transferring the rest of the system components (EDIT, LINK, etc.) to the target device

After step 4 above, the target device may be bootstrapped and the remainder of the build procedure may be carried out while executing the system from the new (and perhaps faster) system device. Because the above build steps involve standard RT-11 file operations, system programs are used for the build procedure. When building from DECTape, PIP is used; from cassette, CBUILD is used; and from paper tape, LINK and PIP are used. RT-11 V02 can be bootstrapped write-protected, and will run PIP write-protected as well. System building should always be carried out with the master write-protected.

### A.1.2 DECTape from DECTape or Disk, and Disk from DECTape or Disk

On DECTape or DECpack disk, RT-11 is distributed as a "ready-to-run" Single-Job system. When bootstrapped (as described in Chapter 2), the system is running and may be used to build other DECTape and disk systems. The following files on the distributed system DECTape or disk have special significance (all are not present on all media):

|            |   |
|------------|---|
| MONITR.SYS | On a DECTape master this is a Single-Job DECTape Monitor; on a DECpack disk master and a Tall cassette master this is a Single-Job RK11 Disk Monitor. When the volume is booted, this file is the monitor used. |
| DTMNSJ.SYS | This is a Single-Job DECTape Monitor, used for building DECTape-based Single-Job systems (not available on paper tape or cassette).   |
| DTMNFB.SYS | This is a Foreground/Background DECTape Monitor, and is used when it is desired to run the F/B system from DECTape (not available on paper tape or cassette).   |
| RKMNSJ.SYS | This is a Single-Job RK11 Monitor. When building a Single-Job disk system, this file becomes MONITR.SYS on the disk.  |
| RKMNFB.SYS | This is a Foreground/Background RK11 Monitor. If an RK11 F/B system is desired, this file becomes MONITR.SYS on the disk.   |



## Assembly, Link, and Build Instructions

|           |   |
|-----------|---|
| RFMNFBSYS | This is a Single-Job RFl1 Monitor, corresponding to RKMNSJ.SYS (not available on cassette).   |
| RFMNFBSYS | This is a Foreground/Background RFl1 Monitor, corresponding to RKMNFBSYS (not available on cassette).   |
| RF.SYS    | This file is an RFl1/RS11 device handler, which allows RT-11 systems running from DECTape or DECpack disks to access the RFl1 disk (not available on cassette). |
| RK.SYS    | This is an RK11 device handler, which allows RT-11 to read and write RK11 disks while running a DECTape or RFl1 system.   |
| DT.SYS    | This is a TC11 DECTape handler, which allows RT-11 to read and write DECTape while running disk systems (not available on DECTape or cassette).                 |
| CR.SYS    | This is the card reader handler (CR11, CM11).   |
| MT.SYS    | This is the magtape handler (TM11/TU10).  |
| CT.SYS    | This file is the cassette (TAl1) handler.   |
| PR.SYS    | This is the high-speed reader (PC-11) handler.  |
| PP.SYS    | This is the high-speed punch (PC-11) handler.   |
| TT.SYS    | This is the general terminal handler.   |
| LP.SYS    | This is the line printer (LP-11, LS-11, LV-11) handler.   |

To build another DECTape or DECpack system (on unit n) from a running DECTape or DECpack system, the following set of commands to PIP can be used (commands illustrated in this appendix are terminated with the carriage return key).

| <u>Commands</u>         | <u>Explanation</u>                   |
|-------------------------|--------------------------------------|
| .R PIP                  |                                      |
| *DKn:/Z                 | Initialize the new DECTape or Disk.  |
| DKn/Z ARE YOU SURE?Y    |                                      |
| *DKn:A=DK0:/S           | Copy all files from DK0 to DKn.      |
| *DKn:A=DKn:MONITR.SYS/U | Write the hardware bootstrap on DKn. |

Unit n now is a ready-to-run copy of the system DECTape or Disk.

In the above, the system files were copied to DK: via the /S option in PIP; in actuality, any method of copying the files to the new device would have sufficed. For example, the command:

\*DKn:A=DK0:/S



## Assembly, Link, and Build Instructions

could have been replaced by:

```
*DKn:*.*=DK0:*.*/Y/X
```

or

```
*DKn:MONITR.SYS=MONITR.SYS/Y
```

```
*DKn:LP.SYS=LP.SYS/Y
```

.

.

etc., until all desired files were transferred.

To build an RK11 disk system from a running DECTape system, use the following commands:

| <u>Commands</u>                  | <u>Explanation</u>  |
|----------------------------------|---|
| .R PIP                           |   |
| *RK:/Z                           | Initialize the disk.  |
| RK:/Z ARE YOU SURE?Y             |   |
| *RK:A=DT0:/S                     | Copy the DECTape files onto disk.   |
| *RK:DTMNSJ.SYS=RK:MONITR.SYS/Y/R | Rename the DECTape monitor on the disk to an appropriate name.                            |
| *RK:MONITR.SYS=RK:RKMNSJ.SYS/Y/R | Rename the disk monitor on the disk to MONITR.SYS (in this case, the Single-Job Monitor). |
| *RK:A=RK:MONITR.SYS/U            | Write the system bootstrap on the disk.   |

The RK11 disk may now be bootstrapped as described in Chapter 2, or with the PIP /O option. The original DECTape MONITR.SYS file must be renamed to another name before the RK monitor is named to MONITR.SYS, or it will be automatically deleted when the RK monitor rename is performed.

To build an RF11 disk system from a running disk or DECTape system, use the following commands:

| <u>Commands</u>                  | <u>Explanation</u>   |
|----------------------------------|--|
| .R PIP                           |  |
| *RF:/Z                           | Initialize the disk.   |
| RF:/Z ARE YOU SURE ?Y            |  |
| *RF:*.*=SY:*.*/X/Y               | Copy all files from the running system to the new system.                                  |
| *RF:XXMNYY.SYS=RF:MONITR.SYS/Y/R | Rename the DECTape or RK monitor on the RF to an appropriate name.                         |
| *RF:MONITR.SYS=RF:RFMNFB.SYS/Y/R | Rename the desired RF monitor on the RF disk to MONITR.SYS (in this case the F/B Monitor). |
| *RF:A=RF:MONITR.SYS/U            | Write the system bootstrap on the RF disk.   |



## Assembly, Link, and Build Instructions

The RFl1 disk may now be bootstrapped as described in Chapter 2, or with the PIP /O option.

Note that in the above examples, all files were copied from the running system to the system being built. Although this is a common practice, it is not necessary, and is usually not possible when the running system is disk and the target device is DECTape. The requirement is that the following elements be transferred:

1. A monitor file
2. The handlers for the desired devices
3. The hardware bootstrap
4. Those programs and files which will be used with the new system

When building a system for a configuration which contains only LA30 and DECTape, it is wise to avoid transferring any of the unrequired handler files (DT is the system handler in the monitor) as they require space on the system DECTape, yet serve no purpose. Users of 8K machines may choose to build systems without the file MACRO.SAV, while EXPAND.SAV and ASEMBL.SAV can be eliminated from systems with 16K or more of memory. The system is distributed with six monitor files, yet the system only requires one for day-to-day operation, and seldom would an application have a need for more than two (the Single Job and Foreground/Background monitors often reside on the same volume for convenient switching).

### A.1.3 Switching Between Single-Job and Foreground/Background

For an application that requires frequent switching between the F/B and Single-job monitors, the following procedure is followed:

1. Both monitors reside on the same volume, the one running named MONITR.SYS and the other called XXXXXX.SYS. (The actual name XXXXXX is not significant.)
2. When a change-over is desired, PIP is used to:
  - a. Preserve the running monitor by renaming it to YYYYYY.SYS
  - b. Rename the desired monitor to MONITR.SYS
  - c. Write the new bootstrap from the new MONITR.SYS file
  - d. Reboot the system

For example, assume an RK11 system is running the Single-Job Monitor, with the Foreground/Background Monitor present as the file RKMNFB.SYS.

|                            |                                  |
|----------------------------|----------------------------------|
| .R PIP                     |                                  |
| *RKMNSJ.SYS=MONITR.SYS/Y/R | Preserve the Single-Job Monitor. |
| ?REBOOT?                   |                                  |
| *MONITR.SYS=RKMNFB.SYS/Y/R | Activate the F/B Monitor by      |
| ?REBOOT?                   | renaming it to MONITR.SYS.       |
| *RK:A=MONITR.SYS/U         | Write the new bootstrap.         |
| *RK:/O                     | Reboot the system.               |



## Assembly, Link, and Build Instructions

### A.1.4 Disk from Cassette

On cassette, RT-11 is distributed as a series of RT-11 files on several cassettes, each cassette labeled DEC-11-ORTSA-C-TCn. The following files on the system cassettes have special significance:

CBUILD.SYS      CBUILD is the special system-build version of PIP which is loaded via the cassette bootstrap. This program is used to initialize the disk and transfer the remaining files from cassette to disk. CBUILD.SYS is used to build RK11 systems only.

MONITR.SYS      The RT-11 Single-Job RK11 Monitor file.

The remaining .SYS files are the monitor and handler files as described for the DECpack disk and DECTape systems in A.1.2.

To build an RT-11 system from cassette, perform the following operations:

1. Mount an RT-11 cassette which has the file CBUILD.SYS as its first file on Unit 0.
2. Bootstrap cassette Unit 0 as follows:

Load and start the system bootstrap loader (called CBOOT). This can be done in one of two ways:

- a. If the system has a hardware cassette bootstrap, enter 173300 in the Switch Register, press LOAD ADDR and START. (Step b may be ignored.)
- b. If no hardware bootstrap is available, CBOOT must be manually loaded and started by the user. Two versions of CBOOT are provided. The standard version is the version used in the hardware bootstrap and consists of the 28 words listed in Table A-1.

A shorter (20-word) version called QCBOOT may optionally be loaded by the user. This version does not provide some of the error checking and handling which the longer CBOOT does, but allows a faster means of manually booting the system. The binary instructions are also listed in Table A-1:



## Assembly, Link, and Build Instructions

Table A-1  
CBOOT (QCBOOT) Instructions

| Location | CBOOT    | QCBOOT   |
|----------|----------|----------|
|          | Contents | Contents |
| 001000   | 012700   | 012700   |
| 001002   | 177500   | 177500   |
| 001004   | 005010   | 005010   |
| 001006   | 010701   | 010701   |
| 001010   | 062701   | 062701   |
| 001012   | 000052   | 000034   |
| 001014   | 012702   | 112102   |
| 001016   | 000375   | 112110   |
| 001020   | 112103   | 032710   |
| 001022   | 112110   | 100240   |
| 001024   | 100413   | 001775   |
| 001026   | 130310   | 100001   |
| 001030   | 001776   | 005007   |
| 001032   | 105202   | 005202   |
| 001034   | 100772   | 100770   |
| 001036   | 116012   | 116012   |
| 001040   | 000002   | 000002   |
| 001042   | 120337   | 000766   |
| 001044   | 000000   | 017775   |
| 001046   | 001767   | 002415   |
| 001050   | 000000   |          |
| 001052   | 000755   |          |
| 001054   | 005710   |          |
| 001056   | 100774   |          |
| 001060   | 005007   |          |
| 001062   | 017640   |          |
| 001064   | 002415   |          |
| 001066   | 112024   |          |

After the bootstrap has been manually loaded (using the Switch Register, LOAD ADDR, and DEP keys), set 001000 in the switches, press LOAD ADDR and START. At this point the RUN lamp should be lit and the system cassette should begin to move.

3. CBUILD will print the following on the console terminal.

CBUILD Version number  
\*

CBUILD is a stand-alone version of PIP. Use the following set of commands (or their equivalent) to build the disk system.

### Commands

### Explanation

\*RK:/Z  
RK:/Z ARE YOU SURE ?Y

Initialize disk directory.

\*RK:\*.\*=CTn:\*.\*/Y/X

Write system files on disk.  
Repeat this command for each cassette necessary to build the desired system.



## Assembly, Link, and Build Instructions

\*RK:A=CTn:MONITR.SYS/U                      Write bootstrap on disk.

\*RK:/O    Boot the disk system.

RT-11 is then running from the disk and may be used in the normal fashion to copy any other desired files from cassette to disk. Note that no devices other than disk or cassette can be used until their handler files have been added to the disk and the system has been rebooted.

### A.1.5 Disk from Paper Tape

RT-11 is distributed as object modules on paper tape. Two of the tapes (PT BUILD Tape 1 and 2) are used to place a rudimentary V01-15 monitor and Linker on the disk. The disk system is then started and the Linker is used to link OLDPIP from paper tape onto the disk. Once linked, OLDPIP is used to copy the remaining paper tapes onto the disk, where they can be linked to complete the system.

The following RT-11 paper tapes have special significance:

|  |   |   |
|--|---|---|
| DEC-11-ORPBA-C-PB1<br>RK PT BUILD Tape 1 | } | This paper tape is for RK11 systems only. |
| DEC-11-ORPBA-C-PB2<br>PT BUILD Tape 2    |   |   |
| DEC-11-ORPBA-C-PB3<br>RF PT BUILD Tape 1 | } | This paper tape is for RF11 systems only. |
|  |   |   |

To build an RT-11 system from paper tape, perform the following operations:

1. Load the Bootstrap Loader at 37744, then use it to load the Absolute Loader.
2. Using the Absolute Loader, load the appropriate PT BUILD Tape 1 for the system device desired. It self-starts and prints:

PT BUILD Version number

3. There is a 10-15 second pause, after which PT BUILD prints:

PLACE SECOND TAPE IN READER.  
STRIKE ANY CHARACTER TO CONTINUE.

4. Place the tape PT BUILD Tape 2 into the reader, then strike any character to start the tape.

There is a slight pause, after which the following is printed.

|                      |                               |
|----------------------|-------------------------------|
| DISK BUILD COMPLETE. | [A rudimentary V01-15 monitor |
| RT-11 V01-15         | is used to build the V02      |
| .                    | system.]                      |



## Assembly, Link, and Build Instructions

During the build procedure, only a

WRITE FAILED

error message has exact meaning. If encountered, check to make sure the system disk is not write protected. Any other error indicates a hardware or disk problem.

### 5. Link OLDPIP as follows:

```
.R LINK
*OLDPIP=PR:
↑↑*
```

For each occurrence of the prompting "↑", place the tape OLDPIP.OBJ (DEC-11-OROPA-C-PR) in the reader, then strike a character to read the tape. Type CTRL C when the second "\*" is printed.

### 6. Run OLDPIP to copy the remaining tapes on the system disk (use the /B switch for all files but SYSMAC.SML, SYSMAC.8K, and VTMAC.MAC, which require /A). Unlike the V02 system when built, OLDPIP prompts with an up-arrow or circumflex prior to reading each tape, and proceeds when a character is struck at the keyboard. For example:

| <u>COMMAND</u>     | <u>USE TAPE</u>                 |
|--------------------|---------------------------------|
| .R OLDPIP          |                                 |
| *PATCH.OBJ=PR:/B   | (PATCH.OBJ DEC-11-ORPAA-C-PR)   |
| ↑*PIP.OBJ=PR:/B    | (PIP.OBJ DEC-11-ORPPA-C-PR)     |
| ↑*ODT.OBJ=PR:/B    | (ODT.OBJ DEC-11-ORODA-C-PR)     |
| ↑*PREXEC.OBJ=PR:/B | (PREXEC.OBJ DEC-11-OREXA-C-PR1) |
| ↑*PREPAS.OBJ=PR:/B | (PREPAS.OBJ DEC-11-OREXA-C-PR2) |
| ↑*SMEXEC.OBJ=PR:/B | (SMEXEC.OBJ DEC-11-ORTAA-C-PR1) |
| ↑*SMMAC.OBJ=PR:/B  | (SMMAC.OBJ DEC-11-ORTAA-C-PR2)  |
| ↑*SMPST.OBJ=PR:/B  | (SMPST.OBJ DEC-11-ORTAA-C-PR3)  |
| ↑*RTEXEC.OBJ=PR:/B | (RTEXEC.OBJ DEC-11-ORMAA-C-PR1) |
| ↑*RTMAC.OBJ=PR:/B  | (RTMAC.OBJ DEC-11-ORMAA-C-PR2)  |
| ↑*RTPST.OBJ=PR:/B  | (RTPST.OBJ DEC-11-ORMAA-C-PR3)  |
| ↑*SYSMAC.SML=PR:/A | (SYSMAC.SML DEC-11-ORSYA-C-PA1) |
| ↑*SYSMAC.8K=PR:/A  | (SYSMAC.8K DEC-11-ORSYA-C-PA2)  |
| ↑*VTMAC.MAC=PR:/A  | (VTMAC.MAC DEC-11-OVTMA-C-PA)   |
| ↑*VTHDLR.OBJ=PR:/B | (VTHDLR.OBJ DEC-11-OVTHA-C-PR)  |
| ↑*CT.OBJ=PR:/B     | (CT.OBJ DEC-11-OCAHA-C-PR)      |
| ↑*MT.OBJ=PR:/B     | (MT.OBJ DEC-11-OMTHA-C-PR)      |
| ↑*CR.OBJ=PR:/B     | (CR.OBJ DEC-11-OCRHA-C-PR)      |
| ↑*PP.OBJ=PR:/B     | (PP.OBJ DEC-11-ORTPA-C-PR)      |
| ↑*PR.OBJ=PR:/B     | (PR.OBJ DEC-11-ORPHA-C-PR)      |
| ↑*LP.OBJ=PR:/B     | (LP.OBJ DEC-11-ORTLA-C-PR)      |
| ↑*TT.OBJ=PR:/B     | (TT.OBJ DEC-11-ORTTA-C-PR)      |
| ↑*RF.OBJ=PR:/B     | (RF.OBJ DEC-11-ORFHA-C-PR)      |
| ↑*RK.OBJ=PR:/B     | (RK.OBJ DEC-11-ORKHA-C-PR)      |
| ↑*RFBTFB.OBJ=PR:/B | (RFBTFB.OBJ DEC-11-ORBTA-C-PR1) |
| ↑*RKBTFB.OBJ=PR:/B | (RKBTFB.OBJ DEC-11-ORBTA-C-PR2) |
| ↑*RFBTSJ.OBJ=PR:/B | (RFBTSJ.OBJ DEC-11-ORBTA-C-PR3) |
| ↑*RKBTSJ.OBJ=PR:/B | (RKBTSJ.OBJ DEC-11-ORBTA-C-PR4) |
| ↑*RT11FB.OBJ=PR:/B | (RT11FB.OBJ DEC-11-ORMNA-C-PR1) |
| ↑*RT11SJ.OBJ=PR:/B | (RT11SJ.OBJ DEC-11-ORMNA-C-PR2) |



## Assembly, Link, and Build Instructions

|                    |             |                     |
|--------------------|-------------|---------------------|
| ↑*EDIT.OBJ=PR:/B   | (EDIT.OBJ   | DEC-11-ORTEA-C-PR1) |
| ↑*VTCED1.OBJ=PR:/B | (VTCED1.OBJ | DEC-11-ORTEA-C-PR2) |
| ↑*VTCED4.OBJ=PR:/B | (VTCED4.OBJ | DEC-11-ORTEA-C-PR3) |
| ↑*VTBEDT.OBJ=PR:/B | (VTBEDT.OBJ | DEC-11-ORTEA-C-PR4) |
| ↑*LINK0.OBJ=PR:/B  | (LINK0.OBJ  | DEC-11-ORLLA-C-PR1) |
| ↑*LNKOV1.OBJ=PR:/B | (LNKOV1.OBJ | DEC-11-ORLLA-C-PR2) |
| ↑*LNKOV2.OBJ=PR:/B | (LNKOV2.OBJ | DEC-11-ORLLA-C-PR3) |
| ↑*LNKOV3.OBJ=PR:/B | (LNKOV3.OBJ | DEC-11-ORLLA-C-PR4) |
| ↑*LNKOV4.OBJ=PR:/B | (LNKOV4.OBJ | DEC-11-ORLLA-C-PR5) |
| ↑*LNKOV5.OBJ=PR:/B | (LNKOV5.OBJ | DEC-11-ORLLA-C-PR6) |
| ↑*DUMP.OBJ=PR:/B   | (DUMP.OBJ   | DEC-11-ORDMA-C-PR)  |
| ↑*SRCCOM.OBJ=PR:/B | (SRCCOM.OBJ | DEC-11-OSRCA-C-PR)  |
| ↑*FILEX.OBJ=PR:/B  | (FILEX.OBJ  | DEC-11-ORFLA-C-PR)  |
| ↑*LIBR0.OBJ=PR:/B  | (LIBR0.OBJ  | DEC-11-ORLBA-C-PR1) |
| ↑*LIBR1.OBJ=PR:/B  | (LIBR1.OBJ  | DEC-11-ORLBA-C-PR2) |
| ↑*LIBR2.OBJ=PR:/B  | (LIBR2.OBJ  | DEC-11-ORLBA-C-PR3) |
| ↑*LIBR3.OBJ=PR:/B  | (LIBR3.OBJ  | DEC-11-ORLBA-C-PR4) |
| ↑*LIBR4.OBJ=PR:/B  | (LIBR4.OBJ  | DEC-11-ORLBA-C-PR5) |
| ↑*CREF.OBJ=PR:/B   | (CREF.OBJ   | DEC-11-ORCFA-C-PR)  |
| ↑*PAT0.OBJ=PR:/B   | (PAT0.OBJ   | DEC-11-ORPOA-C-PR1) |
| ↑*PAT1.OBJ=PR:/B   | (PAT1.OBJ   | DEC-11-ORPOA-C-PR2) |
| ↑*PAT2.OBJ=PR:/B   | (PAT2.OBJ   | DEC-11-ORPOA-C-PR3) |
| ↑*PAT3.OBJ=PR:/B   | (PAT3.OBJ   | DEC-11-ORPOA-C-PR4) |
| ↑*PAT4.OBJ=PR:/B   | (PAT4.OBJ   | DEC-11-ORPOA-C-PR5) |
| ↑*PAT5.OBJ=PR:/B   | (PAT5.OBJ   | DEC-11-ORPOA-C-PR6) |
| ↑*PAT6.OBJ=PR:/B   | (PAT6.OBJ   | DEC-11-ORPOA-C-PR7) |
| ↑*IRAD50.OBJ=PR:/B | (IRAD50.OBJ | DEC-11-ORPOA-C-PR8) |
| ↑*R50ASC.OBJ=PR:/B | (R50ASC.OBJ | DEC-11-ORPOA-C-PR9) |

PREEXEC.OBJ and PREPAS.OBJ are the object modules for EXPAND. SMEXEC.OBJ, SMMAC.OBJ and SMPST.OBJ are linked for ASEMBL, while RTEEXEC.OBJ, RTMAC.OBJ and RTPST.OBJ are the MACRO object modules. PIP.OBJ is the object module for PIP.

VTHDLR, after a pass by the Librarian, becomes the display handler. CT.OBJ through RK.OBJ are linked into handlers of the same name; RFBTFB.OBJ through RT11SJ are the RFl1 and RK11 monitor components. EDIT.OBJ and VTLEDT.OBJ make the editor, while LINK0 through LNKOV5 are the components for the V02 Linker. DUMP.OBJ, SRCCOM.OBJ, FILEX.OBJ, and CREF.OBJ are linked for programs of the same name, while LIBR0 through LIBR4 become the Librarian. PAT0 through R50ASC are linked with RT-11 FORTRAN (if available) to make PATCHO.

If a specific component is not required, it is not necessary to transfer the corresponding tapes.

7. Run LINK to generate the V02 .SYS files and program .SAV files as described later in this appendix (Section A.4.2). Linking the new V02 Linker will generate ADDITIVE REF messages, which should be ignored. Once the V02 Linker is built, it should be used for subsequent linking operations.
8. Reboot the system (with PIP if desired). The .OBJ files (except for ODT and VTLIB) can be deleted as well as OLDP1P.SAV, as they are no longer required.



## Assembly, Link, and Build Instructions

### A.2 CUSTOMIZATION FOR SPECIAL HARDWARE

#### A.2.1 High Baud Rate Serial Console Devices

The serial LA30 requires that filler characters follow each carriage return; the 600, 1200 and 2400 baud VT05's require that filler characters follow each line feed. RT-11 has established a mechanism by which any number of fills may follow any character. The byte at location 56(8) contains the character to be followed by fillers and the byte at location 57(8) contains the number of null fills to be used. These locations are initially set to zero which results in no fillers being generated (normal operation for LT33 and LA30P).

Depending on the terminal, modify the locations as follows:

|                | <u>Loc 56</u> | <u>Loc 57</u> | <u>Resulting Word (octal)</u> |
|----------------|---------------|---------------|-------------------------------|
| LA30s 110 baud | 015(8)        | 002(8)        | 1015                          |
| LA30s 150 baud | 015(8)        | 004(8)        | 2015                          |
| LA30s 300 baud | 015(8)        | 012(8)        | 5015                          |
| VT05 600 baud  | 012(8)        | 001(8)        | 412                           |
| VT05 1200 baud | 012(8)        | 002(8)        | 1012                          |
| VT05 2400 baud | 012(8)        | 004(8)        | 2012                          |

The proper octal word can be changed permanently in the monitor by using PATCH to modify locations 56 and 57 in the monitor file. For example:

| <u>Commands</u>      | <u>Explanation</u>   |
|----------------------|----------------------|
| .R PATCH             |                      |
| PATCH Version number |                      |
| FILE NAME--          |                      |
| *MONITR.SYS/M        |                      |
| *56\0                | 15<LF> Fill after CR |
| *57\0                | 4 with 4 nulls.      |
| *E                   |                      |

Once the change has been made and the bootstrap has been rewritten with the PIP /U switch, all programs which use the monitor for console I/O will operate correctly.

#### A.2.2 7-Track Magtape

The RT-11 magtape handler is distributed such that it will correctly handle both 7- and 9-track magtape without modification. It does so at 800 BPI, using the TM11 dump mode for 7-track drives. 7-track drives can also be written (in hardware mode only; See Appendix H) at 200, 556 and 800 BPI (non-dump mode) by modifying the handler as described below.

The density value is located at 1104(8) in the magtape handler, and can be modified with PATCH. To alter the magtape density used by the handler, simply modify location 1104(8) to the appropriate value from the table below. Note that the density of 9-track tapes cannot be



## Assembly, Link, and Build Instructions

altered. Note also that all drives are written in the same density; if different drives of different densities are to be used on the same machine, different handlers must be used for each.

| <u>Density</u>    | <u>Value of 1104(8)</u> |
|-------------------|-------------------------|
| 200 BPI, 7-TRK    | 000000                  |
| 556 BPI, 7-TRK    | 020000                  |
| 800 BPI, 7-TRK    | 040000                  |
| 800 BPI-DUMP MODE |                         |
| 7-TRK             | 060000                  |
| 800 BPI, 9-TRK    | 060000                  |

For example, to cause RT-11 to write 7-track tapes at 556 BPI:

```
.R PATCH
PATCH Version number

FILE NAME--
*MT.SYS
*1104/60000 20000
*E
```

### A.2.3 Specifying the Number of RF11 Platters

RF-11 is distributed with RF11 support initialized for one RF11 platter. To allow RT-11 to make use of more than one platter, the device size table in the various monitor files must be modified as follows:

| <u>New Value of Table</u> | <u>Number of Platters</u> |
|---------------------------|---------------------------|
| 2000                      | 1                         |
| 4000                      | 2                         |
| 6000                      | 3                         |
| 10000                     | 4                         |

| <u>Monitor</u> | <u>Table Address to Modify</u> |
|----------------|--------------------------------|
| RKMNSJ         | 30702                          |
| RKMNFB         | 32602                          |
| DTMNSJ         | 30702                          |
| DTMNFB         | 32602                          |
| RFMNSJ         | 30702                          |
| RFMNFB         | 32602                          |

For example, to modify the RF11 F/B Monitor for 3 RF/RS11 platters, type:

```
.R PATCH
PATCH Version number

FILE NAME--
*RFMNFB.SYS/M
*32602/2000    6000
*E
```



## Assembly, Link, and Build Instructions

Once the above change has been made, zeroing (PIP /Z switch) the RF disk will adjust the directory to the appropriate size. If the system is already running off RF11 as the system device and the disk cannot be zeroed without destroying the system, compressing the disk (with the PIP /S switch) will automatically re-adjust the directory size.

### A.2.4 Specifying a 50-cycle Clock Rate

RT-11 is distributed with the Keyboard Monitor TIME command calculations based on a 60-cycle clock rate. To cause the TIME command to base calculation on a 50-cycle rate, the monitor is modified such that bit 5 (40(8)) is set in the monitor configuration word (see Section 9.2.6). The address of the CONFIG word in the single job monitors is 33300, and 35300 in the F/B monitors. For example:

For the F/B monitors

.R PATCH

PATCH Version number

FILE NAME--

\*RKMNFB.SYS/M

\*35300/1 41

\*E

For the S/J monitors

.R PATCH

PATCH Version number

FILE NAME--

\*RKMNSJ.SYS/M

\*33300/0 40

\*E

### A.3 SYSTEM OPTIMIZATION

When building RT-11 systems, performance can be optimized (especially on DEctape and DECpack disk) by proper placement of .SYS files on the system device.

Optimal file placement is:

MONITR.SYS

Most frequently used handler

.

.

.

Least frequently used handler

SYSMAC.SML

.

.

(if many assembly operations are performed)



## Assembly, Link, and Build Instructions

Most frequently used program (usually PIP.SAV)

Least frequently used program

Considerations for the above placements are:

1. By positioning all .SYS files at the beginning of the device, movement of .SYS files is avoided during /S operations with PIP, thus eliminating the necessity to reboot.
2. Placement of the monitor immediately following the directory optimizes device motion during monitor swapping operations.
3. Positioning the handlers and programs in descending order related to frequency of use reduces the access times for those files.
4. In systems that will be used for frequent assembly operations, assembler performance can be improved by placing SYSMAC.SML near the beginning of the device.

DECTape users can also conserve time and space by placing only those files needed on the system DECTape. Users of 8K DECTape systems need not place files such as MACRO.SAV and RK.SYS on the system device, as these files cannot be used in 8K systems.

## A.4 ASSEMBLY AND LINKING INSTRUCTIONS

### A.4.1 General Instructions

All RT-11 components, except MACRO, ASEMBL, and the monitors, require 16K of memory to be assembled. MACRO, ASEMBL, and the monitors require 20K. RT-11 MACRO is used as the assembler, and RT-11 LINK is used as the Linker in all cases. All assemblies (except ODT) and all links (except RK.SYS, RF.SYS and DT.SYS) should be error free.

Throughout the following sections, the conventions used are:

1. Default extensions are not explicitly specified. For all the source files, the extensions are .MAC. The assembler output is .OBJ and Linker output is .SAV.
2. The system macro library, SYSMAC.SML, must be on the system device during all assemblies given below.
3. In the example command strings, the sources are kept on logical device SRC:, binary output is to device BIN:, and listing and map files are output to LST:. In actual practice, any appropriate device can be used.
4. The example command strings were executed on a 28K computer and the "FREE CORE" error messages reflect that fact. The actual number of free memory words in each installation will vary, and is not of importance.



## Assembly, Link, and Build Instructions

All RT-11 system assembling and linking operations are normal operations, and the command strings in the descriptions below can be altered to take full advantage of all RT-11 MACRO and LINK command features.

### A.4.2 Assembling and Linking the System Files

The result of the operations below is the six monitors and ten handler files. The UNDEF GLBLS messages resulting from RK.SYS, RF.SYS and DT.SYS linking is expected. The monitor files are named as described in Section A.1.2.

.R MACRO

\*BIN:RT11SJ,LST:RT11SJ=SRC:KMON,USR,RMONSJ,KMOVLY  
ERRORS DETECTED: 0  
FREE CORE: 9943. WORDS

\*BIN:RT11FB,LST:RT11FB=SRC:BFDEF,KMON,USR,RMONFB,KMOVLY  
ERRORS DETECTED: 0  
FREE CORE: 8232. WORDS

\*BIN:RKBTSJ,LST:RKBTSJ=SRC:BSTRAP  
ERRORS DETECTED: 0  
FREE CORE: 15687. WORDS

\*BIN:RFBTSJ,LST:RFBTSJ=SRC:RFSYS,BSTRAP  
ERRORS DETECTED: 0  
FREE CORE: 15679. WORDS

\*BIN:DTBTSJ,LST:DTBTSJ=SRC:DTSYS,BSTRAP  
ERRORS DETECTED: 0  
FREE CORE: 15671. WORDS

\*BIN:RKBTFB,LST:RKBTFB=SRC:BFDEF,BSTRAP  
ERRORS DETECTED: 0  
FREE CORE: 15659. WORDS

\*BIN:RFBTFB,LST:RFBTFB=SRC:RFSYS,BFDEF,BSTRAP  
ERRORS DETECTED: 0  
FREE CORE: 15651. WORDS

\*BIN:DTBTFB,LST:DTBTFB=SRC:DTSYS,BFDEF,BSTRAP  
ERRORS DETECTED: 0  
FREE CORE: 15643. WORDS

\*BIN:RK,LST:RK=SRC:RK  
ERRORS DETECTED: 0  
FREE CORE: 16575. WORDS

\*BIN:RF,LST:RF=SRC:RF  
ERRORS DETECTED: 0  
FREE CORE: 16591. WORDS

\*BIN:DT,LST:DT=SRC:DT  
ERRORS DETECTED: 0  
FREE CORE: 16584. WORDS



Assembly, Link, and Build Instructions

\*BIN:TT,LST:TT=SRC:TT  
ERRORS DETECTED: 0  
FREE CORE: 16562. WORDS

\*BIN:LP,LST:LP=SRC:LP  
ERRORS DETECTED: 0  
FREE CORE: 16564. WORDS

\*BIN:PR,LST:PR=SRC:PR  
ERRORS DETECTED: 0  
FREE CORE: 16670. WORDS

\*BIN:PP,LST:PP=SRC:PP  
ERRORS DETECTED: 0  
FREE CORE: 16663. WORDS

\*BIN:CR,LST:CR=SRC:CR  
ERRORS DETECTED: 0  
FREE CORE: 16201. WORDS

\*BIN:MT,LST:MT=SRC:MT  
ERRORS DETECTED: 0  
FREE CORE: 15564. WORDS

\*BIN:CT,LST:CT=SRC:CT  
ERRORS DETECTED: 0  
FREE CORE: 15631. WORDS

.R LINK

\*BIN:RKMNFB.SYS,LST:RKMNFB=BIN:RKBTFB,RT11FB,RK

\*BIN:RKMNSJ.SYS,LST:RKMNSJ=BIN:RKBTSJ,RT11SJ,RK

\*BIN:RFMNFB.SYS,LST:RFMNFB=BIN:RFBTFB,RT11FB,RF

\*BIN:RFMNSJ.SYS,LST:RFMNSJ=BIN:RFBTSJ,RT11SJ,RF

\*BIN:DTMNFB.SYS,LST:DTMNFB=BIN:DTBTFB,RT11FB,DT

\*BIN:DTMNSJ.SYS,LST:DTMNSJ=BIN:DTBTSJ,RT11SJ,DT

\*BIN:RK.SYS,LST:RK=BIN:RK  
UNDEF GLBLS

\*BIN:RF.SYS,LST:RF=BIN:RF  
UNDEF GLBLS

\*BIN:DT.SYS,LST:DT=BIN:DT  
UNDEF GLBLS

\*BIN:TT.SYS,LST:TT=BIN:TT

\*BIN:LP.SYS,LST:LP=BIN:LP

\*BIN:PR.SYS,LST:PR=BIN:PR

\*BIN:PP.SYS,LST:PP=BIN:PP

\*BIN:CR.SYS,LST:CR=BIN:CR



## Assembly, Link, and Build Instructions

\*BIN:MT.SYS,LST:MT=BIN:MT

\*BIN:CT.SYS,LST:CT=BIN:CT

### A.4.3 Assembling and Linking EDIT

.R MACRO

\*BIN:VTCED1,LST:VTCED1=SRC:EDITDF,VTCAL1

ERRORS DETECTED: 0

FREE CORE: 16654. WORDS

\*BIN:VTCED4,LST:VTCED4=SRC:EDITDF,VTCAL4

ERRORS DETECTED: 0

FREE CORE: 16120. WORDS

\*BIN:VTBEDT,LST:VTBEDT=SRC:EDITDF,VTBASE

ERRORS DETECTED: 0

FREE CORE: 16415. WORDS

\*BIN:EDIT,LST:EDIT=SRC:VTMAC,EDIT

ERRORS DETECTED: 0

FREE CORE: 12873. WORDS

.R LINK

\*BIN:EDIT,LST:EDIT=BIN:VTCED1,VTCED4,VTBEDT,EDIT

### A.4.4 Assembling and Linking MACRO

.R MACRO

\*BIN:RTEEXEC,LST:RTEEXEC=SRC:RTPAR,RPARAM,RCIOCH,RTEEXEC

ERRORS DETECTED: 0

FREE CORE: 13816. WORDS

\*BIN:RTMAC,LST:RTMAC=SRC:RTPAR,RPARAM,RCIOCH,MACRO3,MACRO5

ERRORS DETECTED: 0

FREE CORE: 9174. WORDS

\*BIN:RTPST,LST:RTPST=SRC:RTPAR,PST

ERRORS DETECTED: 0

FREE CORE: 16258. WORDS

.R LINK

\*BIN:MACRO,LST:MACRO=BIN:RTEEXEC,RTMAC,RTPST

### A.4.5 Assembling and Linking EXPAND

.R MACRO

\*BIN:PREEXEC,LST:PREEXEC=SRC:PREPAR,PPARAM,PCIOCH,PREEXEC

ERRORS DETECTED: 0

FREE CORE: 15181. WORDS

\*BIN:PREPAS,LST:PREPAS=SRC:PREPAR,PPARAM,PCIOCH,PREPAS

ERRORS DETECTED: 0

FREE CORE: 14893. WORDS

.R LINK

\*BIN:EXPAND,LST:EXPAND=BIN:PREEXEC,PREPAS



## Assembly, Link, and Build Instructions

### A.4.6 Assembling and Linking ASEMBL

.R MACRO

\*BIN:SMEEXEC,LST:SMEEXEC=SRC:SMPAR,RPARAM,RCIOCH,RTEEXEC

ERRORS DETECTED: 0

FREE CORE: 14059. WORDS

\*BIN:SMMAC,LST:SMMAC=SRC:SMPAR,RPARAM,RCIOCH,MACRO3,MACRO5

ERRORS DETECTED: 0

FREE CORE: 9921. WORDS

\*BIN:SMPST,LST:SMPST=SRC:SMPAR,PST

ERRORS DETECTED: 0

FREE CORE: 16254. WORDS

.R LINK

\*BIN:ASEMBL,LST:ASEMBL=BIN:SMEEXEC,SMMAC,SMPST

### A.4.7 Assembling and Linking CREF

.R MACRO

\*BIN:CREF,LST:CREF=SRC:CREF

ERRORS DETECTED: 0

FREE CORE: 14792. WORDS

.R LINK

\*BIN:CREF,LST:CREF=BIN:CREF

### A.4.8 Assembling and Linking LINK

.R MACRO

\*BIN:LINK0,LST:LINK0=SRC:LINK0

ERRORS DETECTED: 0

FREE CORE: 13944. WORDS

\*BIN:LNKOV1,LST:LNKOV1=SRC:LNKOV1

ERRORS DETECTED: 0

FREE CORE: 14583. WORDS

\*BIN:LNKOV2,LST:LNKOV2=SRC:LNKOV2

ERRORS DETECTED: 0

FREE CORE: 14834. WORDS

\*BIN:LNKOV3,LST:LNKOV3=SRC:LNKOV3

ERRORS DETECTED: 0

FREE CORE: 14530. WORDS

\*BIN:LNKOV4,LST:LNKOV4=SRC:LNKOV4

ERRORS DETECTED: 0

FREE CORE: 14726. WORDS

\*BIN:LNKOV5,LST:LNKOV5=SRC:LNKOV5

ERRORS DETECTED: 0

FREE CORE: 14434. WORDS



## Assembly, Link, and Build Instructions

```
.R LINK
*BIN:LINK,LST:LINK=BIN:LINK0/C
*BIN:LNKOV1/O:1/C
*BIN:LNKOV2/O:1/C
*BIN:LNKOV3/O:1/C
*BIN:LNKOV4/O:1/C
*BIN:LNKOV5/O:1
```

### A.4.9 Assembling and Linking LIBR

```
.R MACRO
*BIN:LIBR0,LST:LIBR0=SRC:LIBR0
ERRORS DETECTED: 0
FREE CORE: 14480. WORDS
```

```
*BIN:LIBR1,LST:LIBR1=SRC:LIBR1
ERRORS DETECTED: 0
FREE CORE: 14635. WORDS
```

```
*BIN:LIBR2,LST:LIBR2=SRC:LIBR2
ERRORS DETECTED: 0
FREE CORE: 14827. WORDS
```

```
*BIN:LIBR3,LST:LIBR3=SRC:LIBR3
ERRORS DETECTED: 0
FREE CORE: 15068. WORDS
```

```
*BIN:LIBR4,LST:LIBR4=SRC:LIBR4
ERRORS DETECTED: 0
FREE CORE: 14996. WORDS
```

```
.R LINK
*BIN:LIBR,LST:LIBR=BIN:LIBR0/C
*BIN:LIBR1/O:1/C
*BIN:LIBR2/O:1/C
*BIN:LIBR3/O:1/C
*BIN:LIBR4/O:1
```

### A.4.10 Assembling and Linking PIP

```
.R MACRO
*BIN:PIP,LST:PIP=SRC:PIP
ERRORS DETECTED: 0
FREE CORE: 13664. WORDS
```

```
.R LINK
*BIN:PIP,LST:PIP=BIN:PIP
```

### A.4.11 Assembling and Linking FILEX

```
.R MACRO
*BIN:FILEX,LST:FILEX=SRC:FILEX
ERRORS DETECTED: 0
FREE CORE: 12986. WORDS
```

```
.R LINK
*BIN:FILEX,LST:FILEX=BIN:FILEX
```



## Assembly, Link, and Build Instructions

### A.4.12 Assembling and Linking SRCCOM

```
.R MACRO
*BIN:SRCCOM,LST:SRCCOM=SRC:SRCCOM
ERRORS DETECTED: 0
FREE CORE: 15293. WORDS
```

```
.R LINK
*BIN:SRCCOM,LST:SRCCOM=BIN:SRCCOM
```

### A.4.13 Assembling and Linking DUMP

```
.R MACRO
*BIN:DUMP,LST:DUMP=SRC:DUMP
ERRORS DETECTED: 0
FREE CORE: 15597. WORDS
```

```
.R LINK
*BIN:DUMP,LST:DUMP=BIN:DUMP
```

### A.4.14 Assembling and Linking PATCH

```
.R MACRO
*BIN:PATCH,LST:PATCH=SRC:PATCH
ERRORS DETECTED: 0
FREE CORE: 15524. WORDS
```

```
.R LINK
*BIN:PATCH,LST:PATCH=BIN:PATCH
```

### A.4.15 Compiling and Linking PATCHO

PATCHO is written in FORTRAN, and as such requires RT-11 FORTRAN IV for compilation and linking.

```
.R FORTRA
*BIN:PAT0,LST:PAT0=SRC:PAT0/S/P/N:5/R:120
*BIN:PAT1,LST:PAT1=SRC:PAT1/S/P
*BIN:PAT2,LST:PAT2=SRC:PAT2/S/P
*BIN:PAT3,LST:PAT3=SRC:PAT3/S/P
*BIN:PAT4,LST:PAT4=SRC:PAT4/S/P
*BIN:PAT5,LST:PAT5=SRC:PAT5/S/P
*BIN:PAT6,LST:PAT6=SRC:PAT6/S/P
```

```
.R MACRO
*BIN:IRAD50,LST:IRAD50=SRC:IRAD50
ERRORS DETECTED: 0
FREE CORE: 16732. WORDS
```

```
*BIN:R50ASC,LST:R50ASC=SRC:R50ASC
ERRORS DETECTED: 0
FREE CORE: 16720. WORDS
```



## Assembly, Link, and Build Instructions

The following linking instructions require that the FORTRAN library, FORLIB.OBJ, be available on the system device. FORLIB.OBJ is available as part of the RT-11 FORTRAN Kit.

```
.R LINK
*BIN:PATCHO=BIN:PAT0/I/C/B:1100/F
*BIN:PAT1,IRAD50/O:1/C
*BIN:PAT3,R50ASC/O:1/C
*BIN:PAT2/O:2/C
*BIN:PAT4/O:2/C
*BIN:PAT5/O:2/C
*BIN:PAT6/O:2
```

```
LIBRARY SEARCH:
$SHORT
```

### A.4.16 Assembling ODT

ODT assembles with one error (a "z" error which flags an ODT instruction which is machine dependent). The error is necessary and should be ignored.

```
.R MACRO
*BIN:ODT,LST:ODT=SRC:ODT
ERRORS DETECTED: 1
FREE CORE: 15730. WORDS
```

### A.4.17 Assembling and Building the VT11 Display Handler Library

```
.R MACRO
*BIN:VTCAL1,LST:VTCAL1=SRC:VTCAL1
ERRORS DETECTED: 0
FREE CORE: 16658. WORDS
```

```
*BIN:VTCAL2,LST:VTCAL2=SRC:VTCAL2
ERRORS DETECTED: 0
FREE CORE: 16726. WORDS
```

```
*BIN:VTCAL3,LST:VTCAL3=SRC:VTCAL3
ERRORS DETECTED: 0
FREE CORE: 16693. WORDS
```

```
*BIN:VTCAL4,LST:VTCAL4=SRC:VTCAL4
ERRORS DETECTED: 0
FREE CORE: 16140 WORDS
```

```
*BIN:VTBASE,LST:VTBASE=SRC:VTBASE
ERRORS DETECTED: 0
FREE CORE: 16355. WORDS
```

```
.R PIP
*BIN:VTHDLR.OBJ=BIN:VTCAL1.OBJ,VTCAL2.OBJ,VTCAL3.OBJ,VTCAL4.OBJ,VTBASE.OBJ/B
```

```
.R LIBR
*BIN:VTLIB=BIN:VTHDLR
```







## APPENDIX B

### COMMAND AND SWITCH SUMMARIES

Command and switch summaries of the various RT-11 system and utility programs are grouped here for the user's convenience. Refer to the appropriate chapter for details.

#### B.1 KEYBOARD MONITOR (Chapter 2)

##### B.1.1 Command Summary

Only those command characters underlined need be entered; all command lines are terminated by typing a carriage return.

| <u>Command Format</u>                        | <u>Used Under</u>   | <u>Explanation</u>   |
|--|---------------------|--|
| <u>ASSIGN</u> dev:udev                       | F/B,S/J             | Assigns a user-defined name (udev) as an alternate name for a device (dev). Deassigns synonyms when used without any arguments.  |
| <u>B</u> location                            | F/B,S/J             | Sets a relocation base (location), which is an octal address to be used as a base address for subsequent Examine and Deposit commands.   |
| <u>CLOSE</u>                                 | F/B,S/J<br>(B only) | Causes all currently open files to become permanent.   |
| <u>DATE</u> dd-mmm-yy                        | F/B,S/J             | Enters the indicated day-month-year (dd-mmm-yy); this date is then assigned to newly created files, new device directory entries, and listing output. When used without an argument, the current date (as entered) is printed. |
| <u>D</u> location = value1,value2,...,valuen | F/B,S/J             | Deposits the specified values starting at the given location (location represents an octal address which is added to the base address to obtain the actual address at which values will be deposited).                         |



## Command and Switch Summaries

| <u>Command Format</u>                | <u>Used Under</u>   | <u>Explanation</u>  |
|--------------------------------------|---------------------|---|
| <u>E</u> location m-location n       | F/B,S/J             | Prints the contents of the specified locations in octal on the console terminal (location represents an octal address which is added to the base address to obtain the actual address examined).  |
| <u>FRUN</u> dev:filnam.ext/N:n/S:n/P | F/B<br>(F only)     | Initiates a foreground job which exists on the device indicated (dev) under the specified filename and extension. /N:n is optionally used to allocate n (decimal) words over and above actual program size; /S:n is optionally used to allocate n words for stack space; /P is used for debugging purposes (the load address is printed but the program must be started using RSUME).           |
| <u>GET</u> dev:filnam.ext            | F/B,S/J             | Loads the specified memory image file (filnam.ext) into memory from the indicated device (dev:).  |
| <u>GT OFF</u>                        | F/B,S/J             | Used (after GT ON) to clear the display processor and resume printout on the console terminal.  |
| <u>GT ON</u> /L:n/T:n                | F/B,S/J             | Enables the display processor so that the display screen replaces the console as the terminal output device. /L:n may be optionally used to designate the number of lines to display (12" screen - 1<=n<=37 octal; 17" screen - 1<=n<=50 octal). /T:n may be optionally used to indicate the top position of the scroll display (12" screen - 1<=n<=1350 octal; 17" screen - 1<=n<=1750 octal). |
| <u>INITIALIZE</u>                    | F/B,S/J<br>(B only) | Resets background system tables; makes nonresident all handlers not loaded and purges background's I/O channels.  |
| <u>LOAD</u> dev,...                  | F/B,S/J             | Makes a device handler resident for use.  |
| <u>LOAD</u> dev=B,dev=F,...          | F/B                 | Makes a device handler resident for use with background and foreground jobs.  |
| <u>R</u> filnam.ext                  | F/B,S/J<br>(B only) | Loads the specific memory image file (filnam.ext) into memory from the system device and starts execution.  |
| <u>REENTER</u>                       | F/B,S/J             | Starts a program at its reentry address (i.e., its start address -2).   |
| <u>RSUME</u>                         | F/B<br>(F only)     | Resumes execution of a foreground program where it was suspended.   |
| <u>RUN</u> dev:filnam.ext            | F/B,S/J<br>(B only) | Loads the specified memory image file (filnam.ext) into memory from the indicated device (dev:) and starts execution.   |



## Command and Switch Summaries

| <u>Command Format</u>           | <u>Used Under</u>            | <u>Explanation</u>   |
|---------------------------------|------------------------------|--|
| <u>SAVE</u> dev:filnam.ext      | areal,area2-arean<br>F/B,S/J | Writes the area(s) of user memory specified into the named file (filnam.ext) in save image format. Memory is transferred in 256-word blocks.               |
| <u>SET</u> dev:{NO}option=value | F/B,S/J                      | Used to change device (dev:) handler characteristics and certain system configuration parameters. See Table 2-5 (Section 2.7.2.8) for a list of options.   |
| <u>START</u> address            | F/B,S/J                      | Begins execution of the program currently in memory at the specified address. If an address is not indicated, the starting address in location 40 is used. |
| <u>SUSPEND</u>                  | F/B<br>(F only)              | Suspends execution of the foreground job currently running.  |
| <u>TIME</u> hh:mm:ss            | F/B,S/J                      | Enters time of day in hours, minutes, seconds past midnight (hh:mm:ss). If all three arguments are omitted, the current time of day is output.             |
| <u>UNLOAD</u> dev,dev,...       | F/B,S/J                      | Makes previously loaded handlers (dev) nonresident and frees the memory space they were using.   |

### B.1.2 Special Function Keys

| <u>Key</u> | <u>Used Under</u> | <u>Function</u>   |
|------------|-------------------|---|
| CTRL A     | F/B,S/J           | Valid when the monitor GT ON command has been typed and the display is in use. Does not echo on the terminal. Used after CTRL S has been typed to effectively page output.  |
| CTRL B     | F/B               | Echoes B> on the terminal and causes all keyboard input to be directed to the background job. At least one line of output will be taken from the background job (the foreground job has priority and control will revert to it if it has output). B> does not echo if output is already coming from the background job. (Control can be redirected to the foreground job via CTRL F.) |



## Command and Switch Summaries

| <u>Key</u> | <u>Used Under</u> | <u>Function</u>   |
|------------|-------------------|---|
| CTRL C     | F/B,S/J           | Echoes ↑C on the terminal, interrupts current program execution, and returns control to the Keyboard Monitor. If a program is waiting for terminal input or is using the device handler TT: for input, typing a single CTRL C interrupts execution and returns control to the monitor command level. Otherwise, two CTRL C's must be typed in order to interrupt execution.                       |
| CTRL E     | F/B,S/J           | Valid when the monitor GT ON command has been typed and the display is in use. Does not echo on the terminal, but causes all I/O to appear on both the display screen and the console terminal simultaneously. A second CTRL E disables console terminal output.  |
| CTRL F     | F/B               | Echoes F> on the terminal and directs all keyboard input to the foreground job and all output to be taken from the foreground job. Control remains with the foreground job until redirected to the background job (via CTRL B) or until the foreground job terminates.  |
| CTRL O     | F/B,S/J           | Echoes ↑O on the terminal and causes suppression of teleprinter output while continuing program execution. Teleprinter output is reenabled when one of the following occurs: <ol style="list-style-type: none"> <li>1. A second CTRL O is typed</li> <li>2. A return to the monitor is indicated via CTRL C</li> <li>3. The running program issues a .RCTRLO directive (see Chapter 9)</li> </ol> |
| CTRL Q     | F/B,S/J           | Does not echo. Resumes printing characters on the terminal from the point at which printing was previously stopped (via CTRL S).  |
| CTRL S     | F/B,S/J           | Does not echo. Temporarily suspends output to the terminal until a CTRL Q is typed. If GT ON is in effect, each subsequent CTRL A causes output to proceed until the screen has been refilled once.   |
| CTRL U     | F/B,S/J           | Echoes ↑U followed by a carriage return on the terminal and deletes the current input line.   |
| CTRL Z     | F/B,S/J           | Echoes ↑Z on the terminal and terminates input when used with the terminal device handler (TT:).  |



## Command and Switch Summaries

| <u>Key</u> | <u>Used Under</u> | <u>Function</u>  |
|------------|-------------------|--|
| RUBOUT     | F/B,S/J           | Deletes the last character from the current line. Echoes a backslash plus the character deleted; each succeeding RUBOUT deletes and echoes another character; an enclosing backslash is printed when a key other than RUBOUT is typed. |

### B.2 EDITOR (Chapter 4)

#### B.2.1 Command Arguments

| <u>Format</u> | <u>Meaning</u>   |
|---------------|--|
| n             | A decimal integer (in the range -16383 to +16383) which may, except where noted, be preceded by a + or -. Whenever an argument is acceptable in a command, its absence implies an argument of 1. |
| 0             | Refers to the beginning of the current line.   |
| /             | Refers to the end of the text in the current Text Buffer.  |
| =             | Is used with the J, D and C commands only and represents -n, where n is equal to the length of the last text argument used.  |

#### B.2.2 Input and Output Commands

| <u>Command</u> | <u>Form</u>            | <u>Meaning</u>   |
|----------------|------------------------|--|
| EDIT BACKUP    | EB dev:filnam.ext[n]\$ | Opens a file for editing, creating a backup copy (.BAK).                                       |
| EDIT READ      | ER dev:filnam.ext\$    | Opens a file for input.  |
| EDIT WRITE     | EW dev:filnam.ext[n]\$ | Creates a new file for output.   |
| END FILE       | EF                     | Closes the current output file without performing any further input/output operations.         |
| EXIT           | EX                     | Outputs the remainder of the input file to the output file and returns control to the monitor. |
| LIST           | (-)nL<br>0L<br>/L      | Prints a specified number of lines on the console terminal.                                    |



## Command and Switch Summaries

| <u>Command</u> | <u>Form</u>       | <u>Meaning</u>   |
|----------------|-------------------|--|
| NEXT           | nN                | Outputs the contents of the Text Buffer to the output file, clears the buffer, and reads in the next page of the input file. |
| READ           | R                 | Reads a page of text from the input file and appends it to the contents of the buffer.                                       |
| VERIFY         | V                 | Prints the current text line (the line containing the pointer) on the console terminal.                                      |
| WRITE          | (-)nW<br>0W<br>/W | Outputs a specified number of lines of text from the Text Buffer to the output file.   |

### B.2.3 Pointer Relocation Commands

| <u>Command</u> | <u>Form</u>             | <u>Meaning</u>   |
|----------------|-------------------------|--|
| ADVANCE        | (-)nA<br>0A<br>/A       | Moves the pointer over a specified number of lines in the Text Buffer. The pointer is positioned at the beginning of the line. |
| BEGINNING      | B                       | Moves the current location pointer to the beginning of the Text Buffer.  |
| JUMP           | (-)nJ<br>0J<br>/J<br>=J | Moves the pointer over a specified number of characters in the Text Buffer.  |

### B.2.4 Search Commands

| <u>Command</u> | <u>Form</u> | <u>Meaning</u>  |
|----------------|-------------|---|
| FIND           | nFtext\$    | Beginning at the current location pointer, searches the entire text file for the nth occurrence of the specified character string. Pages of text are read into the Text Buffer, searched, and then written to the output file until the text string is found. |
| GET            | nGtext\$    | Searches the contents of the Text Buffer, beginning at the current location pointer, for the next occurrence of the text string.  |
| POSITION       | nPttext\$   | Searches the input file for the nth occurrence of the text string; if the text string is not found, the buffer is cleared and a new page is read from the input file.   |



## Command and Switch Summaries

### B.2.5 Text Modification Commands

| <u>Command</u> | <u>Form</u>                         | <u>Meaning</u>  |
|----------------|-------------------------------------|---|
| CHANGE         | (-)nCtext\$<br>0Ctext\$<br>/Ctext\$ | Replaces n characters, beginning at the pointer, with the indicated text string.                          |
| DELETE         | (-)nD<br>0D<br>/D<br>=D             | Removes a specified number of characters from the Text Buffer, beginning at the current location pointer. |
| EXCHANGE       | (-)nXtext\$<br>0Xtext\$<br>/Xtext\$ | Replaces n lines, beginning at the pointer, with the indicated text string.                               |
| INSERT         | Itext\$                             | Inserts text immediately following the current location pointer; an ALTMODE terminates the text.          |
| KILL           | (-)nK<br>0K<br>/K                   | Removes n lines from the Text Buffer beginning at the current location pointer.                           |

### B.2.6 Utility Commands

| <u>Command</u> | <u>Form</u>                    | <u>Meaning</u>  |
|----------------|--------------------------------|---|
| EDIT CONSOLE   | EC                             | If scroller is in use, EC returns scroller to its normal mode (using full screen for display of text and commands); if scroller is not in use, EC clears screen and returns control to console terminal (following ED). |
| EDIT DISPLAY   | ED                             | If scroller is in use, ED recalls the text window (following EC) and arranges text and commands on screen; if scroller is not in use, ED displays text window only.   |
| EXECUTE MACRO  | nE                             | Executes the command string specified in the last macro command.  |
| MACRO          | M/command string/<br>OM<br>M// | Inserts a command string into the Macro Buffer.<br>Clears the Macro Buffer and reclaims the area for text.  |
| SAVE           | nS                             | Copies the specified number of lines, beginning at the pointer, into the Save Buffer.   |
| UNSAVE         | U                              | Inserts the entire contents of the Save Buffer into the Text Buffer at the position of the current location pointer.  |



## Command and Switch Summaries

| <u>Command</u> | <u>Form</u> | <u>Meaning</u>   |
|----------------|-------------|--|
| EDIT VERSION   | EV          | Displays the version number of the Editor on the console terminal. |

### B.2.7 Immediate Mode Commands

| <u>Command</u>  | <u>Meaning</u>  |
|---|---|
| CTRL N  | Advances the pointer (cursor) to the beginning of the next line.                  |
| CTRL G  | Moves the pointer (cursor) to the beginning of the previous line.                 |
| CTRL D  | Moves the pointer (cursor) forward by one character.                              |
| CTRL V  | Moves the pointer (cursor) back by one character.                                 |
| RUBOUT  | Deletes the character immediately preceding the pointer (cursor).                 |
| ALTMODE (two)<br>(one only)   | Enters Immediate Mode.<br>Returns control to Editor Command Mode.                 |
| Any character other<br>than the above (with<br>the exception of CTRL C) | Inserts the character as text positioned immediately before the pointer (cursor). |

### B.2.8 Key Commands

| <u>Command</u> | <u>Meaning</u>  |
|----------------|---|
| ALTMODE        | Echoes \$. A single ALTMODE terminates a text string. A double ALTMODE executes the command string. (When used alone on a line, two ALTMODES cause control to enter Immediate Mode, while a single ALTMODE returns control to Editor Command Mode.) |
| CTRL C         | Echoes at the terminal as ↑C and a carriage return. Terminates execution of EDIT commands, closes any open files, and returns to monitor command mode.  |
| CTRL O         | Echoes ↑O and a carriage return. Inhibits printing on the terminal until completion of the current command string. Typing a second CTRL O resumes output.   |
| CTRL U         | Echoes ↑U and a carriage return. Deletes all the characters on the current terminal input line.   |



## Command and Switch Summaries

| <u>Command</u> | <u>Meaning</u>   |
|----------------|--|
| RUBOUT         | Deletes character from the current line.   |
| TAB            | Spaces to the next tab stop. Tab stops are positioned every eight spaces on the terminal.  |
| CTRL X         | Echoes ↑X and a carriage return. CTRL X causes the Editor to ignore the entire command string currently being entered. The Editor prints a <CR><LF> and an asterisk to indicate that the user may enter another command. |

### B.3 PIP (Chapter 4)

#### B.3.1 Switch Summary

| <u>Switch</u>   | <u>Explanation</u>   |
|-----------------|--|
| /A              | Copies file(s) in ASCII mode; ignores nulls and rubouts; converts to 7-bit ASCII.  |
| /B              | Copies files in formatted binary mode.   |
| /C              | Used in conjunction with another switch; causes only files with current date to be included in the specified operation.  |
| /D              | Deletes file(s) from specified device.   |
| /E              | Lists the device directory including unused spaces and their sizes. Sequence numbers are listed for cassettes.   |
| /F              | Prints a short directory (filenames only) of the specified device.   |
| /G              | Ignores any input errors which occur during a file transfer and continues copying.   |
| /I or no switch | Copies file(s) in image mode (byte by byte).   |
| /K              | Scans the specified device and types the absolute block numbers (in octal) of any bad blocks on the device.  |
| /L              | Lists the directory of the specified device. Sequence numbers are listed for cassettes.  |
| /M:n            | Used when I/O transfers involve either cassette or magtape, n represents the numeric position of the file to be accessed in relation to the physical position of the cassette or magtape on the drive. |



## Command and Switch Summaries

| <u>Switch</u> | <u>Explanation</u>   |
|---------------|--|
| /N:n          | Used with /Z to specify the number of directory blocks (n) to allocate to the directory.   |
| /O            | Bootstraps the specified device (DT0, RKn, or RF only).  |
| /Q            | Causes PIP to type each filename which is eligible for a wild card operation and to ask for a confirmation of its inclusion in the operation.  |
| /R            | Renames the specified file.  |
| /S            | Compresses the files on the specified directory device so that free blocks are combined into one area.   |
| /T            | Extends number of blocks allocated for a file.   |
| /U            | Copies the bootstrap from the specified file into absolute blocks 0 and 2 of the specified device.   |
| /V            | Types the version number of the PIP program being used.  |
| /W            | Includes the absolute starting block and any extra directory words in the directory listing for each file on the device (numbers in octal). Used with /F, /L, or /E.                                 |
| /X            | Copies files individually (without concatenation).   |
| /Y            | Causes system files and .BAD files to be operated on by the command specified.   |
| /Z:n          | Zeroes (initializes) the directory of the specified device; n is used to allocate extra words per directory entry. When used with /N, the number of directory segments for entries may be specified. |

### B.4 MACRO/CREF (Chapter 5)

Refer to Appendix C for a complete summary of MACRO features. CREF switches are also included in that appendix.



## Command and Switch Summaries

### B.5 LINKER (Chapter 6)

#### B.5.1 Switch Summary

The Linker switches (and the command line on which each must appear) are:

| <u>Switch Name</u> | <u>Command Line</u> | <u>Meaning</u>  |
|--------------------|---------------------|---|
| /A                 | 1st                 | Alphabetizes the entries in the load map.   |
| /B:n               | 1st                 | Bottom address of program is indicated as n (illegal for foreground links).   |
| /C                 | any                 | Continues input files on another command line (must be used with /O).   |
| /F                 | 1st                 | Indicates that the Linker will use the default FORTRAN library, FORLIB.OBJ.   |
| /I                 | 1st                 | Includes the global symbols to be searched from the library.  |
| /L                 | 1st                 | Produces an output file in LDA format (illegal for foreground links).   |
| /M:n               | 1st                 | Allows terminal keyboard specification of the user's stack address. n represents an optional 6-digit unsigned octal number.   |
| /O:n               | any but the 1st     | Indicates that the program will be an overlay structure; n specifies the overlay region to which the module is assigned.  |
| /R                 | 1st                 | Produces output in REL format; only files in REL format will run in the foreground (REL format files may not be run under a Single-Job system).   |
| /S                 | 1st                 | Allows the maximum amount of space in memory to be available for the Linker's symbol table. (This switch should only be used when a particular link stream causes a symbol table overflow.) |
| /T or /T:n         | 1st                 | Transfer address is to be specified at terminal keyboard via n.   |



## Command and Switch Summaries

### B.6 LIBRARIAN (Chapter 7)

#### B.6.1 Switch Summary

The Librarian (LIBR) switches (and the command line on which each must appear) are:

| <u>Switch</u> | <u>Command<br/>Line</u> | <u>Meaning</u>  |
|---------------|-------------------------|---|
| /C            | Any                     | The command is too long for the current line and is continued on the next line. |
| /D            | 1st                     | Deletes modules from a library file.  |
| /G            | 1st                     | Global deletion; deletes entry points from the library directory.               |
| /R            | 1st                     | Replaces modules in a library file.   |
| /U            | 1st                     | Update; inserts and replaces modules in a library file.                         |

### B.7 ODT (Chapter 8)

#### B.7.1 Command Summary

In the command format shown below, r represents a relocatable expression and n represents an octal number.

| <u>Command</u> | <u>Format</u> | <u>Explanation</u>   |
|----------------|---------------|--|
| RETURN         |               | Closes open location and accepts the next command.   |
| LINE FEED      |               | Closes current location and opens next sequential location.  |
| ↑ or ^         | ↑ or ^        | Opens previous location.   |
| ← or _         | ← or _        | Indexes the contents of the opened location by the contents of the PC and opens the resulting location.      |
| >              | >             | Uses the contents of the opened location as a relative branch instruction and opens the referenced location. |
| <              | <             | Returns to sequence prior to last @, >, or ← command and opens the succeeding location.                      |
| @              | @             | Uses the contents of the opened location as an absolute address and opens that location.                     |



# Command and Switch Summaries

| <u>Command</u> | <u>Format</u> | <u>Explanation</u>  |
|----------------|---------------|---|
| /              | /             | Reopens the last opened location.   |
|                | r/            | Opens the word at location r.   |
| \              | \             | Reopens the last opened byte (SHIFT L).   |
|                | r\            | Opens the byte at location r.   |
| !              | !<br>n!       | After a word or byte has been opened, prints the address of the opened location relative to relocation register n. If n is omitted, ODT selects the relocation register whose contents are closest to but less than or equal to the address of the opened location. |
| \$             | \$n/          | Opens general register n (0-7).   |
|                | \$B/          | Opens the first word of the breakpoint table.   |
|                | \$C/          | Opens Constant Register.  |
|                | \$F/          | Opens Format Register.  |
|                | \$P/          | Opens Priority Register.  |
|                | \$R/          | Opens first Relocation Register (register 0).   |
|                | \$S/          | Opens Status Register.  |
| A              | r;nA          | Starting at location r, prints n bytes in their ASCII format; then inputs n bytes from the terminal starting at location r.   |
| B              | ;B            | Removes all Breakpoints.  |
|                | r;B           | Sets Breakpoint at location r.  |
|                | r;nB          | Sets Breakpoint n at location r.  |
|                | ;nB           | Removes the nth Breakpoint.   |
| C              | r;C           | Prints the value of r and stores it in the Constant Register.   |
| E              | r;E           | Searches for instructions that reference effective address r.   |
| F              | ;F            | Fills memory words with contents of the Constant Register.  |
| G              | r;G           | Goes to location r and starts program.  |
| I              | ;I            | Fills memory bytes with the low-order 8 bits of the Constant Register.  |
| O              | r;O           | Calculates offset from currently open location to r.  |
| P              | ;P            | Proceeds with program execution from breakpoint. In single instruction mode only, executes next instruction.  |



## Command and Switch Summaries

| <u>Command</u> | <u>Format</u> | <u>Explanation</u>  |
|----------------|---------------|---|
|                | k;P           | Proceeds with program execution from breakpoint; stops after encountering the breakpoint k times. In single instruction mode only, executes next k instructions.  |
| R              | ;R            | Sets all Relocation Registers to -1 (highest address value).  |
|                | ;nR           | Sets Relocation Register n to -1.   |
|                | r;nR          | Sets Relocation Register n to the value of r. If n is omitted, it is assumed to be 0.   |
|                | R             | Selects the Relocation Register whose contents are closest to but less than or equal to contents of the opened location. Subtracts the contents of the register from the contents of the opened word and prints the result. |
|                | nR            | Subtracts the contents of the Relocation Register n from the contents of the opened word and prints the result.   |
| S              | ;S            | Disables single instruction mode; reenables breakpoints.  |
|                | ;nS           | Enables single instruction mode (n can have any value and is not significant); disables breakpoints.  |
| W              | r;W           | Searches for words with bit patterns which match r.   |
| X              | X             | Performs a Radix 50 unpack of the binary contents of the current opened word; then permits the storage of a new Radix 50 binary number in the same location.  |

### B.8 PROGRAMMED REQUESTS (Chapter 9)

Appendix E summarizes the programmed requests available under RT-11, Version 2.

### B.9 DUMP (Appendix I)

#### B.9.1 Switch Summary

| <u>Switch</u> | <u>Meaning</u>          |
|---------------|-------------------------|
| /B            | Outputs octal bytes.    |
| /E:n          | Ends output at block n. |



## Command and Switch Summaries

| <u>Switch</u> | <u>Meaning</u>               |
|---------------|------------------------------|
| /G            | Ignores input errors.        |
| /N            | Suppresses ASCII output.     |
| /O:n          | Outputs only block number n. |
| /S:n          | Starts output with block n.  |
| /W            | Outputs octal words.         |
| /X            | Outputs RAD50 characters.    |

### B.10 FILEX (Appendix J)

#### B.10.1 Switch Summary

| <u>Switch</u> | <u>Meaning</u>  |
|---------------|---|
| /A            | Indicates a character-by-character ASCII transfer in which rubouts and nulls are deleted; when /T is also used, each PDP-10 word is assumed to contain five 7-bit ASCII bytes.  |
| /D            | Deletes the named file from the device; valid only for DOS/BATCH and RSTS-11 DECTape.   |
| /F            | Causes a "fast" listing of the device directory by listing filenames only.  |
| /I            | Performs an image mode transfer; if the input is either DOS/BATCH, RSTS-11 or RT-11, this is a word-for-word transfer; if the input is from DECsystem-10, /I indicates that the file resembles a file created on DECsystem-10 by MACY11, MACX11, or LNKX11 with the /I switch: in this case, each DECsystem-10 36-bit word contains one PDP-11 8-bit byte in its low-order bits.  |
| /L            | Causes a complete listing of the device directory, including filenames, block lengths, and creation dates.  |
| /P            | Performs a packed image transfer; if the input is from either DOS/BATCH, RSTS-11 or RT-11, this is a word-for-word transfer; if the input is from DECsystem-10, /P indicates that the file resembles a file created on DECsystem-10 by MACY11, MACX11, or LNKX11 with the /P switch, in which case each DECsystem-10 36-bit word contains four PDP-11 8-bit bytes aligned on bits 0, 8, 18, and 26. This mode is assumed if no mode switch (/A, /I) is indicated in a command line. |



## Command and Switch Summaries

| <u>Switch</u> | <u>Meaning</u>  |
|---------------|---|
| /S            | Indicates the device is a DOS/BATCH (or RSTS-11) file-structured device.  |
| /T            | Indicates the device is a DECsystem-10 file-structured device.  |
| /V            | Types out version number of FILEX.  |
| /Z            | Zeroes the directory of the specified device in the proper format (valid only for DOS/BATCH and RSTS-11 DECTape). |

### B.11 SRCCOM (Appendix K)

#### B.11.1 Switch Summary

| <u>Switch</u> | <u>Meaning</u>  |
|---------------|---|
| /B            | Compares blank lines. Without this switch, blank lines are ignored.   |
| /C            | Ignores comments (all text on a line preceded by a semicolon) and spacing (spaces and tabs).  |
| /F            | Includes form feeds in the output file (form feeds are still compared if /F is not used, but they are not included in the output of differences). |
| /H            | Types list of switches available (help text).   |
| /L:n          | Specifies the number of lines that determine a match (where n is an octal number <=310). The default value for n is 3.                            |
| /S            | Ignores spaces and tabs.  |

### B.12 PATCH (Appendix L)

#### B.12.1 Command Summary

| <u>Command</u> | <u>Meaning</u>                            |
|----------------|---|
| /O             | Indicates overlay-structured file.        |
| /M             | Indicates monitor file.                   |
| Vr;nR          | Sets relocation register n to value Vr.   |
| b:B            | Sets bottom address of overlay file to b. |



## Command and Switch Summaries

| <u>Command</u> | <u>Meaning</u>   |
|----------------|--|
| [s:]r,o/       | Opens word location Vr + o in overlay segment s.                   |
| [s:]r,o\       | Opens byte location Vr + o in overlay segment s.                   |
| <CR>           | Closes currently open word/byte.                                   |
| <LF>           | Closes currently open word/byte and opens the next one.            |
| ↑ or ^         | Closes the currently open word/byte and opens the previous one.    |
| @              | Closes the currently open word and opens the word addressed by it. |
| F              | Begins patching a new file.  |
| E              | Exits to RT-11 monitor.  |

## B.13 PATCHO (Appendix M)

### B.13.1 Command Summary

| <u>Command</u> | <u>Format</u>   | <u>Meaning</u>  |
|----------------|---|---|
| BYTE           | BYTE CSECT + OFFSET = $\left\{ \begin{array}{l} \# \text{ NAME OF} \\ \% \text{ CSECT OR} \\ \text{GLOBAL} \end{array} \right\} \left\{ \begin{array}{l} + \\ - \end{array} \right\} \text{OFFSET}$<br>NAME | Modifies a given byte in an object module.  |
| DEC            | *DEC  | Used when the proper checksum for the patch being made is unknown. PATCHO computes a checksum and prints out its value.                               |
| DUMP           | *DUMP   | Prints the contents of an object module in octal and causes an automatic POINT to the next module, if any, in the input file.                         |
| EXIT           | *EXIT   | Terminates the patch session by closing the file and returning control to the monitor.  |
| HELP           | *HELP   | Prints an explanation of the PATCHO commands.   |
| LIST           | *LIST   | Lists the names of all object modules in the input file in the order in which they appear in the file. (A POINT command should be used after a LIST.) |
| OPEN           | *OPEN   | Opens files for input and output.   |



# Command and Switch Summaries

| <u>Command</u> | <u>Format</u>  | <u>Meaning</u>  |
|----------------|--|---|
| POINT          | *POINT modnam  | Locates the specified object module (modnam) and prepares it for subsequent WORD, BYTE, or DUMP operations. |
| WORD           | WORD CSECT + OFFSET = $\left\{ \begin{array}{l} \# \\ \% \end{array} \right\} \left\{ \begin{array}{l} \text{NAME OF} \\ \text{CSECT OR} \\ \text{GLOBAL} \end{array} \right\} \left\{ \begin{array}{l} + \\ - \end{array} \right\} \text{OFFSET}$ | Modifies a given word in an object module.  |



## APPENDIX C

### MACRO ASSEMBLER, INSTRUCTION, AND CHARACTER CODE SUMMARIES

#### C.1 ASCII CHARACTER SET

| <u>Even<br/>Parity<br/>Bit</u> | <u>7-Bit<br/>Octal<br/>Code</u> | <u>Character</u> | <u>Remarks</u>   |
|--------------------------------|---------------------------------|------------------|--|
| 0                              | 000                             | NUL              | Null, Tape Feed, CTRL SHIFT P.   |
| 1                              | 001                             | SOH              | Start of Heading; also SOM (Start of Message), CTRL A.                                 |
| 1                              | 002                             | STX              | Start of Text; also EOA (End Of Address), CTRL B.                                      |
| 0                              | 003                             | ETX              | End of Text; also EOM (End Of Message), CTRL C.  |
| 1                              | 004                             | EOT              | End of Transmission (END); Shuts off TWX machines, CTRL D.                             |
| 0                              | 005                             | ENQ              | Enquiry (ENQRY); also WRU, CTRL E.   |
| 0                              | 006                             | ACK              | Acknowledge; also RU, CTRL F.  |
| 1                              | 007                             | BEL              | Rings the Bell. CTRL G.  |
| 1                              | 010                             | BS               | Backspace; also FEO, Format Effector. Backspaces some machines, CTRL H.                |
| 0                              | 011                             | HT               | Horizontal TAB. CTRL I.  |
| 0                              | 012                             | LF               | Line Feed or Line Space (New Line); Advances paper to next line; duplicated by CTRL J. |
| 1                              | 013                             | VT               | Vertical TAB (VTAB). CTRL K.   |
| 0                              | 014                             | FF               | FORM FEED to top of next page (PAGE). CTRL L.  |
| 1                              | 015                             | CR               | Carriage Return to beginning of line. Duplicated by CTRL M.                            |
| 1                              | 016                             | SO               | Shift Out; Changes ribbon color to red. CTRL N.  |
| 0                              | 017                             | SI               | Shift In; Changes ribbon color to black. CTRL O.                                       |
| 1                              | 020                             | DLE              | Data Link Escape. CTRL B (DC0).  |
| 0                              | 021                             | DC1              | Device Control 1, turns transmitter (reader) on, CTRL Q (X ON).                        |
| 0                              | 022                             | DC2              | Device Control 2, turns punch or auxiliary on, CTRL R (TAPE, AUX ON).                  |
| 1                              | 023                             | DC3              | Device Control 3, turns transmitter (reader) off, CTRL S (X OFF).                      |
| 0                              | 024                             | DC4              | Device Control 4, turns punch or auxiliary off, CTRL T (AUX OFF).                      |
| 1                              | 025                             | NAK              | Negative Acknowledge; also ERR, Error, CTRL U.   |



# MACRO Assembler, Instruction, and Character Code Summaries

|   |     |     |                                      |  |
|---|-----|-----|--------------------------------------|--|
| 1 | 026 | SYN | Synchronous File (SYNC), CTRL V.     |  |
| 0 | 027 | ETB | End of Transmission Block; also LEM, |  |
|   |     |     | Logical End of Medium, CTRL W.       |  |
| 0 | 030 | CAN | Cancel (CANCL), CTRL X.              |  |
| 1 | 031 | EM  | End of Medium, CTRL Y.               |  |
| 1 | 032 | SUB | Substitute, CTRL Z.                  |  |
| 0 | 033 | ESC | Escape, CTRL SHIFT K.                |  |
| 1 | 034 | FS  | File Separator, CTRL SHIFT L.        |  |
| 0 | 035 | GS  | Group Separator, CTRL SHIFT M.       |  |
| 0 | 036 | RS  | Record Separator, CTRL SHIFT N.      |  |
| 1 | 037 | US  | Unit Separator, CTRL SHIFT O.        |  |
| 1 | 040 | SP  | Space.                               |  |
| 0 | 041 | !   |                                      |  |
| 0 | 042 | "   |                                      |  |
| 1 | 043 | #   |                                      |  |
| 0 | 044 | \$  |                                      |  |
| 1 | 045 | %   |                                      |  |
| 0 | 046 | &   |                                      |  |
| 0 | 047 | '   | Apostrophe or Acute Accent.          |  |
| 0 | 050 | (   |                                      |  |
| 0 | 051 | )   |                                      |  |
| 1 | 052 | *   |                                      |  |
| 0 | 053 | +   |                                      |  |
| 1 | 054 | ,   |                                      |  |
| 0 | 055 | -   |                                      |  |
| 0 | 056 | .   |                                      |  |
| 1 | 057 | /   |                                      |  |
| 0 | 060 | 0   |                                      |  |
| 1 | 061 | 1   |                                      |  |
| 1 | 062 | 2   |                                      |  |
| 0 | 063 | 3   |                                      |  |
| 1 | 064 | 4   |                                      |  |
| 0 | 065 | 5   |                                      |  |
| 0 | 066 | 6   |                                      |  |
| 1 | 067 | 7   |                                      |  |
| 1 | 070 | 8   |                                      |  |
| 0 | 071 | 9   |                                      |  |
| 0 | 072 | :   |                                      |  |
| 1 | 073 | ;   |                                      |  |
| 0 | 074 | <   |                                      |  |
| 1 | 075 | =   |                                      |  |
| 1 | 076 | >   |                                      |  |
| 0 | 077 | ?   |                                      |  |
| 1 | 100 | @   |                                      |  |
| 0 | 101 | A   |                                      |  |
| 0 | 102 | B   |                                      |  |
| 1 | 103 | C   |                                      |  |
| 0 | 104 | D   |                                      |  |
| 1 | 105 | E   |                                      |  |
| 1 | 106 | F   |                                      |  |
| 0 | 107 | G   |                                      |  |
| 0 | 110 | H   |                                      |  |
| 1 | 111 | I   |                                      |  |
| 1 | 112 | J   |                                      |  |
| 0 | 113 | K   |                                      |  |
| 1 | 114 | L   |                                      |  |
| 0 | 115 | M   |                                      |  |
| 0 | 116 | N   |                                      |  |
| 1 | 117 | O   |                                      |  |
| 0 | 120 | P   |                                      |  |
| 1 | 121 | Q   |                                      |  |



# MACRO Assembler, Instruction, and Character Code Summaries

|   |     |     |  |
|---|-----|-----|--|
| 1 | 122 | R   |  |
| 0 | 123 | S   |  |
| 1 | 124 | T   |  |
| 0 | 125 | U   |  |
| 0 | 126 | V   |  |
| 1 | 127 | W   |  |
| 1 | 130 | X   |  |
| 0 | 131 | Y   |  |
| 0 | 132 | Z   |  |
| 1 | 133 | [   | SHIFT K.   |
| 0 | 134 | \   | SHIFT L.   |
| 1 | 135 | ]   | SHIFT M.   |
| 1 | 136 | ↑   | (Appears as ^ on some machines).   |
| 0 | 137 | ←   | (Appears as _ (Underscore) on some machines).  |
| 0 | 140 | `   | Accent Grave.  |
| 1 | 141 | a   |  |
| 1 | 142 | b   |  |
| 0 | 143 | c   |  |
| 1 | 144 | d   |  |
| 0 | 145 | e   |  |
| 0 | 146 | f   |  |
| 1 | 147 | g   |  |
| 1 | 150 | h   |  |
| 0 | 151 | i   |  |
| 0 | 152 | j   |  |
| 1 | 153 | k   |  |
| 0 | 154 | l   |  |
| 1 | 155 | m   |  |
| 1 | 156 | n   |  |
| 0 | 157 | o   |  |
| 1 | 160 | p   |  |
| 0 | 161 | q   |  |
| 0 | 162 | r   |  |
| 1 | 163 | s   |  |
| 0 | 164 | t   |  |
| 1 | 165 | u   |  |
| 1 | 166 | v   |  |
| 0 | 167 | w   |  |
| 0 | 170 | x   |  |
| 1 | 171 | y   |  |
| 1 | 172 | z   |  |
| 0 | 173 | {   |  |
| 1 | 174 |     |  |
| 0 | 175 | }   |  |
| 0 | 176 | ~   |  |
| 1 | 177 | DEL | This Code Generated by ALTMODE.<br>This Code Generated by PREFIX key (if Present)<br>DELETE, RUBOUT. |

## C.2 RADIX-50 CHARACTER SET

| Character | ASCII Octal Equivalent | Radix-50 Equivalent |
|-----------|------------------------|---------------------|
| space     | 40                     | 0                   |
| A-Z       | 101-132                | 1-32                |



# MACRO Assembler, Instruction, and Character Code Summaries

|        |       |       |
|--------|-------|-------|
| \$     | 44    | 33    |
| .      | 56    | 34    |
| unused |       | 35    |
| 0-9    | 60-71 | 36-47 |

The maximum Radix-50 value is, thus:

$$47 \times 50^2 + 47 \times 50 + 47 = 174777$$

The following table provides a convenient means of translating between the ASCII character set and its Radix-50 equivalents. For example, given the ASCII string X2B, the Radix-50 equivalent is (arithmetic is performed in octal):

X=113000  
2=002400  
B=000002  
            
X2B=115402

| Single Char.<br>or<br>First Char. | Second<br>Character | Third<br>Character |
|-----------------------------------|---------------------|--------------------|
| A                                 | 000050              | A 000001           |
| B                                 | 000120              | B 000002           |
| C                                 | 000170              | C 000003           |
| D                                 | 000240              | D 000004           |
| E                                 | 000310              | E 000005           |
| F                                 | 000360              | F 000006           |
| G                                 | 000430              | G 000007           |
| H                                 | 000500              | H 000010           |
| I                                 | 000550              | I 000011           |
| J                                 | 000620              | J 000012           |
| K                                 | 000670              | K 000013           |
| L                                 | 000740              | L 000014           |
| M                                 | 001010              | M 000015           |
| N                                 | 001060              | N 000016           |
| O                                 | 001130              | O 000017           |
| P                                 | 001200              | P 000020           |
| Q                                 | 001250              | Q 000021           |
| R                                 | 001320              | R 000022           |
| S                                 | 001370              | S 000023           |
| T                                 | 001440              | T 000024           |
| U                                 | 001510              | U 000025           |
| V                                 | 001560              | V 000026           |
| W                                 | 001630              | W 000027           |
| X                                 | 001700              | X 000030           |
| Y                                 | 001750              | Y 000031           |
| Z                                 | 002020              | Z 000032           |
| \$                                | 002070              | \$ 000033          |
| .                                 | 002140              | . 000034           |
|                                   | 002210              | 000035             |
| 0                                 | 002260              | 0 000036           |
| 1                                 | 002330              | 1 000037           |
| 2                                 | 002400              | 2 000040           |



## MACRO Assembler, Instruction, and Character Code Summaries

|   |        |   |        |   |        |
|---|--------|---|--------|---|--------|
| 3 | 147100 | 3 | 002450 | 3 | 000041 |
| 4 | 152200 | 4 | 002520 | 4 | 000042 |
| 5 | 155300 | 5 | 002570 | 5 | 000043 |
| 6 | 160400 | 6 | 002640 | 6 | 000044 |
| 7 | 163500 | 7 | 002710 | 7 | 000045 |
| 8 | 166600 | 8 | 002760 | 8 | 000046 |
| 9 | 171700 | 9 | 003030 | 9 | 000047 |

### C.3 MACRO SPECIAL CHARACTERS

| Character       | Function   |
|-----------------|--|
| form feed       | Source line terminator, forces a new listing page          |
| line feed       | Source line terminator                                     |
| carriage return | Formatting character                                       |
| vertical tab    | Source line terminator                                     |
| :               | Label terminator   |
| =               | Direct assignment indicator                                |
| %               | Register term indicator                                    |
| tab             | Item terminator, field terminator                          |
| space           | Item terminator, field terminator                          |
| #               | Immediate expression indicator                             |
| @               | Deferred addressing indicator                              |
| (               | Initial register indicator                                 |
| )               | Terminal register indicator                                |
| , (comma)       | Operand field separator                                    |
| ;               | Comment field indicator                                    |
| +               | Arithmetic addition operator or autoincrement indicator    |
| -               | Arithmetic subtraction operator or autodecrement indicator |
| *               | Arithmetic multiplication operator                         |
| /               | Arithmetic division operator                               |
| &               | Logical AND operator                                       |
|                 | Logical OR operator  |
| "               | Double ASCII character indicator                           |
| ' (apostrophe)  | Single ASCII character indicator                           |
| .               | Assembly location counter                                  |
| <               | Initial argument indicator                                 |
| >               | Terminal argument indicator                                |
| †               | Universal unary operator                                   |
| †               | Argument indicator   |
| \               | MACRO numeric argument indicator                           |

### C.4 ADDRESS MODE SYNTAX

In the following syntax table, n represents an integer between 0 and 7; R is a register expression; E represents any expression; ER represents either a register expression or an absolute expression in the range 0 to 7.

|    |                   |           |   |
|----|-------------------|-----------|---|
| On | Register          | R         | Register R contains the operand.<br>R is a register expression. |
| ln | Deferred Register | @R or (R) | Register R contains the operand address.                        |



## MACRO Assembler, Instruction, and Character Code Summaries

|    |  |        |   |
|----|--|--------|---|
| 2n | Autoincrement                            | (ER)+  | The contents of the register specified by ER are incremented after being used as the address of the operand.          |
| 3n | Deferred Autoincrement                   | @(ER)+ | ER contains a pointer to the address of the operand. ER is incremented after use.                                     |
| 4n | Autodecrement                            | -(ER)  | The contents of register ER are decremented before being used as the address of the operand.                          |
| 5n | Deferred Autodecrement                   | @-(ER) | The contents of register ER are decremented before being used as a pointer to the address of the operand.             |
| 6n | Index by the Register Specified          | E(ER)  | The value obtained when E is combined with the contents of the register specified (ER) is the address of the operand. |
| 7n | Deferred index by the Register Specified | @E(ER) | E added to ER produces a pointer to the address of the operand.   |
| 27 | Immediate Operand                        | #E     | E is the operand.   |
| 37 | Absolute address                         | @#E    | E is the operand address.   |
| 67 | Relative address                         | E      | E is the address of the operand.  |
| 77 | Deferred relative address                | @E     | E is a pointer to the address of the operand.   |

### C.5 INSTRUCTIONS

The tables of instructions which follow are grouped according to the operands they take and according to the bit patterns of their op-codes.

The following symbols are used to indicate the instruction type format:

|    |                                    |
|----|------------------------------------|
| OP | Instruction mnemonic               |
| R  | Register Expression                |
| E  | Expression                         |
| ER | Register expression or expression  |
|    | 0<=ER<=7                           |
| AC | Floating point register expression |
| A  | General address specification      |



## MACRO Assembler, Instruction, and Character Code Summaries

In the representation of op-codes, the following symbols are used:

|    |                            |                                   |
|----|----------------------------|-----------------------------------|
| SS | Source operand             | Specified by a 6-bit address mode |
| DD | Destination operand        | Specified by a 6-bit address mode |
| XX | 8-bit offset to a location | Branch instructions               |
| R  | Integer between 0 and 7    | Represents a general register     |

Symbols used in the description of instruction operations are:

|     |  |
|-----|--|
| SE  | Source Effective Address               |
| FSE | Floating Source Effective Address      |
| DE  | Destination Effective Address          |
| FDE | Floating Destination Effective Address |
|     | Absolute Value of                      |
| ( ) | Contents of                            |
| →   | Becomes                                |

The condition codes in the processor status word (PS) are affected by the instructions; these condition codes are represented as follows:

|   |               |   |
|---|---------------|---|
| N | Negative bit: | Set if the result is negative           |
| Z | Zero bit:     | Set if the result is zero               |
| V | Overflow bit: | Set if the operation caused an overflow |
| C | Carry bit:    | Set if the operation caused a carry     |

In the representation of the instruction's effect on the condition codes, the following symbols are used:

|   |                   |
|---|-------------------|
| * | Conditionally set |
| - | Not affected      |
| 0 | Cleared           |
| 1 | Set               |

To set conditionally means to use the instruction's result to determine the state of the code.

Logical operators are represented by the following symbols:

|   |  |
|---|--|
|   | Inclusive OR   |
| ⊕ | Exclusive OR   |
| & | AND  |
| — | Used over a symbol to represent the 1's complement of the symbol |



# MACRO Assembler, Instruction, and Character Code Summaries

## C.5.1 Double Operand Instructions (OP A,A)

| Op-Code          | Mnemonic     | Stands for                  | Operation  | Status Word Condition Codes |   |   |   |
|------------------|--------------|-----------------------------|--|-----------------------------|---|---|---|
|                  |              |                             |  | N                           | Z | V | C |
| 01SSDD<br>11SSDD | MOV<br>MOVEB | MOVE<br>MOVE Byte           | (SE) $\rightarrow$ (DE)  | *                           | * | 0 | - |
| 02SSDD<br>12SSDD | CMP<br>CMPB  | CoMPare<br>CoMPare Byte     | (SE) - (DE)  | *                           | * | * | * |
| 03SSDD<br>13SSDD | BIT<br>BITB  | BiT Test<br>BiT Test Byte   | (SE) & (DE)  | *                           | * | 0 | - |
| 04SSDD<br>14SSDD | BIC<br>BICB  | BiT Clear<br>BiT Clear Byte | $\overline{(SE)}$ & (DE) $\rightarrow$ (DE)                      | *                           | * | 0 | - |
| 05SSDD<br>15SSDD | BIS<br>BISB  | BiT Set<br>BiT Set Byte     | (SE) ! (DE) $\rightarrow$ (DE)                                   | *                           | * | 0 | - |
| 06SSDD<br>16SSDD | ADD<br>SUB   | ADD<br>SUBtract             | (SE) + (DE) $\rightarrow$ (DE)<br>(DE) - (SE) $\rightarrow$ (DE) | *                           | * | * | * |

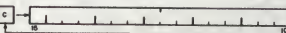

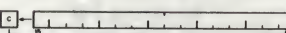

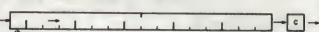
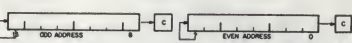
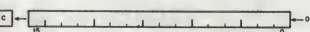
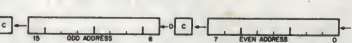
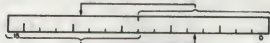
## C.5.2 Single Operand Instructions (OP A)

| Op-code          | Mnemonic    | Stands for                            | Operation                                  | Status Word Condition Codes |   |   |   |
|------------------|-------------|---------------------------------------|--|-----------------------------|---|---|---|
|                  |             |                                       |  | N                           | Z | V | C |
| 0050DD<br>1050DD | CLR<br>CLRB | CLeaR<br>CLeaR Byte                   | 0 $\rightarrow$ (DE)                       | 0                           | 1 | 0 | 0 |
| 0051DD<br>1051DD | COM<br>COMB | COMplement<br>COMplement Byte         | $\overline{(DE)}$ $\rightarrow$ (DE)       | *                           | * | 0 | 1 |
| 0052DD<br>1052DD | INC<br>INCB | INCrement<br>INCrement Byte           | (DE) + 1 $\rightarrow$ (DE)                | *                           | * | * | 1 |
| 0053DD<br>1053DD | DEC<br>DECB | DECrement<br>DECrement Byte           | (DE) - 1 $\rightarrow$ (DE)                | *                           | * | * | - |
| 0054DD<br>1054DD | NEG<br>NEGB | NEGate<br>NEGate Byte                 | $\overline{(DE)}$ + 1 $\rightarrow$ (DE)   | *                           | * | * | * |
| 0055DD<br>1055DD | ADC<br>ADCB | ADd Carry<br>ADd Carry Byte           | $\overline{(DE)}$ + (C) $\rightarrow$ (DE) | *                           | * | * | * |
| 0056DD<br>1056DD | SBC<br>SBCB | SuBtract Carry<br>SuBtract Carry Byte | (DE) - (C) $\rightarrow$ (DE)              | *                           | * | * | * |
| 0057DD<br>1057DD | TST<br>TSTB | TeST<br>TeST Byte                     | (DE)                                       | *                           | * | 0 | 0 |



# MACRO Assembler, Instruction, and Character Code Summaries

## C.5.3 Rotate/Shift

| Op-Code | Mnemonic | Stands for                  | Operation  | Status Word Condition Codes |   |   |   |
|---------|----------|-----------------------------|--|-----------------------------|---|---|---|
|         |          |                             |  | N                           | Z | V | C |
| 0060DD  | ROR      | ROtate Right                |    | *                           | * | * | * |
| 1060DD  | RORB     | ROtate Right Byte           |    | *                           | * | * | * |
| 0061DD  | ROL      | ROtate Left                 |    | *                           | * | * | * |
| 1061DD  | ROLB     | ROtate Left Byte            |    | *                           | * | * | * |
| 0062DD  | ASR      | Arithmetic Shift Right      |    | *                           | * | * | * |
| 1062DD  | ASRB     | Arithmetic Shift Right Byte |    | *                           | * | * | * |
| 0063DD  | ASL      | Arithmetic Shift Left       |   | *                           | * | * | * |
| 1063DD  | ASLB     | Arithmetic Shift Left Byte  |  | *                           | * | * | * |
| 0001DD  | JMP      | JuMP                        | DE → (PC)  | -                           | - | - | - |
| 0003DD  | SWAB     | SWAp Bytes                  |  | *                           | * | 0 | 0 |

The following instructions are available on the PDP-11/35,40,45 as noted:

| Op-Code                              | Mnemonic | Stands for                           | Operation   | Status Word Condition Codes |   |   |   |
|--------------------------------------|----------|--------------------------------------|---|-----------------------------|---|---|---|
|                                      |          |                                      |   | N                           | Z | V | C |
| <u>11/35, 11/40, 11/45 with KT11</u> |          |                                      |   |                             |   |   |   |
| 0065SS                               | MFPI     | Move From Previous Instruction space | (SE) → (TEMP)<br>(SP) - 2 → (SP)<br>(TEMP) → ((SP)) | *                           | * | 0 | - |
| <u>11/45 with KT11 only</u>          |          |                                      |   |                             |   |   |   |
| 1065SS                               | MFPD     | Move From Previous Data space        | (SE) → (TEMP)<br>(SP) - 2 → (SP)<br>(TEMP) → ((SP)) | *                           | * | 0 | - |



# MACRO Assembler, Instruction, and Character Code Summaries

## 11/35, 11/40, 11/45 with KT11 .

|        |      |   |   |         |
|--------|------|---|---|---------|
| 0066DD | MTPI | Move To<br>Previous<br>Instruction<br>space | ((SP)) → (TEMP)<br>(SP+2) → (SP)<br>(TEMP) → (DE) | * * 0 - |
|--------|------|---|---|---------|

## 11/45 with KT11 only

|        |      |                                   |   |         |
|--------|------|-----------------------------------|---|---------|
| 1066DD | MTPD | Move To<br>Previous<br>Data space | ((SP)) → (TEMP)<br>(SP+2) → (SP)<br>(TEMP) → (DE) | * * 0 - |
|--------|------|-----------------------------------|---|---------|

|        |       |                            |          |         |
|--------|-------|----------------------------|----------|---------|
| 1701DD | LDFPS | Load FPP<br>program status | DE → FPS | - - - - |
|--------|-------|----------------------------|----------|---------|

## 11/35, 11/40, 11/45

|        |     |             |   |         |
|--------|-----|-------------|---|---------|
| 0067DD | SXT | Sign eXtend | 0 → DE if N bit<br>is clear<br>-1 → DE if N bit<br>is set | - * - - |
|--------|-----|-------------|---|---------|

## 11/45 with FP11-B

|        |       |   |  |         |
|--------|-------|---|--|---------|
| 0707DD | NEGD  | NEGate Double                                       | -(FDE) → FDE   |         |
| 0707DD | NEGF  | NEGate Floating                                     | -(FDE) → FDE   | * * 0 0 |
| 1702DD | STFPS | STore Floating<br>Point processor<br>program Status | See Chapter 7<br>in PDP-11/45<br>Processor<br>Handbook | - - - - |
| 1703DD | STST  | STore floating<br>point processor<br>STatus         |  | - - - - |
| 1704DD | CLRD  | CLear Double  | 0 → (FDE)  | 0 1 0 0 |
| 1704DD | CLRf  | CLear Floating                                      | 0 → (FDE)  | 0 1 0 0 |
| 1705DD | TSTD  | TeST Double   | (FDE)  | * * 0 0 |
| 1705DD | TSTF  | TeST Floating                                       | (FDE)  | * * 0 0 |
| 1706DD | ABSD  | make ABSolute<br>Double                             | (FDE)  → (FDE)   | 0 * 0 0 |
| 1706DD | ABSF  | make ABSolute<br>Floating                           | (FDE)  → (FDE)   | 0 * 0 0 |



# MACRO Assembler, Instruction, and Character Code Summaries

## C.5.4 Operate Instructions (OP)

| Op-Code | Mnemonic | Stands for                               | Operation  | Status Word Condition Codes |   |   |   |
|---------|----------|--|--|-----------------------------|---|---|---|
|         |          |  |  | N                           | Z | V | C |
| 000000  | HALT     | HALT                                     | The computer stops all functions.  | -                           | - | - | - |
| 000001  | WAIT     | WAIT                                     | The computer stops and waits for an interrupt.   | -                           | - | - | - |
| 000002  | RTI      | ReTurn from Interrupt (Return from Trap) | The PC and ST are popped off the SP stack:<br>(SP) → (PC)<br>(SP)+2 → (SP)<br>((SP)) → (ST)<br>(SP)+2 → (SP) | *                           | * | * | * |
| 000005  | RESET    | RESET                                    | Returns all I/O devices to power-on state.   | -                           | - | - | - |
| 000241  | CLC      | CLear Carry bit                          | 0 → C  | -                           | - | - | 0 |
| 000261  | SEC      | SEt Carry bit                            | 1 → C  | -                           | - | - | 1 |
| 000242  | CLV      | CLear oVerflow                           | 0 → V  | -                           | - | 0 | - |
| 000262  | SEV      | SEt oVerflow bit                         | 1 → V  | -                           | - | 1 | - |
| 000244  | CLZ      | CLear Zero bit                           | 0 → Z  | -                           | 0 | - | - |
| 000264  | SEZ      | SEt Zero bit                             | 1 → Z  | -                           | 1 | - | - |
| 000250  | CLN      | CLear Negative bit                       | 0 → N  | 0                           | - | - | - |
| 000270  | SEN      | SEt Negative bit                         | 1 → N  | 1                           | - | - | - |
| 000257  | CCC      | Clear all Condition Codes                | 0 → N<br>0 → Z<br>0 → V<br>0 → C   | 0                           | 0 | 0 | 0 |
| 000277  | SCC      | Set all Condition Codes                  | 1 → N<br>1 → Z<br>1 → V<br>1 → C   | 1                           | 1 | 1 | 1 |
| 000240  | NOP      | No OPERATION                             |  | -                           | - | - | - |



# MACRO Assembler, Instruction, and Character Code Summaries

The following instructions are available on the PDP-11/45 with FP11-B only:

| Op-Code | Mnemonic | Stands for                   | Operation  | Status Word Condition Codes |   |   |   |
|---------|----------|------------------------------|--|-----------------------------|---|---|---|
|         |          |                              |  | N                           | Z | V | C |
| 170000  | CFCC     | Copy Floating Condition Code | Copy FPP condition codes into CPU condition codes. | *                           | * | * | * |
| 170011  | SETD     | SET Double floating mode     | FPP set to double precision                        | -                           | - | - | - |
| 170001  | SETF     | SET Floating mode            | FPP set to single precision mode                   | -                           | - | - | - |
| 170002  | SETI     | SET Integer mode             | FPP set for integer data (16 bits)                 | -                           | - | - | - |
| 170012  | SETL     | SET Long integer mode        | FPP set for long integer data (32 bits)            | -                           | - | - | - |

## All 11/45's, with and without FP11-B

|        |     |                       |   |   |   |   |   |
|--------|-----|-----------------------|---|---|---|---|---|
| 000006 | RTT | ReTurn from inTerrupt | Same as RTI instruction but inhibits trace trap | * | * | * | * |
|--------|-----|-----------------------|---|---|---|---|---|

## C.5.5 Trap Instructions (OP or OP e where 0<=E<=377(8)) \*(OP (only))

| Op-Code | Mnemonic | Stands for        | Operation                                      | Status Word Condition Codes |   |   |   |
|---------|----------|-------------------|--|-----------------------------|---|---|---|
|         |          |                   |  | N                           | Z | V | C |
| *000003 | BPT      | BreakPoint Trap   | Trap to location 14. This is used to call ODT. | *                           | * | * | * |
| *000004 | IOT      | Input Output Trap | Trap to location 20. This is used to call IOX. | *                           | * | * | * |



# MACRO Assembler, Instruction, and Character Code Summaries

|                   |      |                  |   |         |
|-------------------|------|------------------|---|---------|
| 104000-<br>104377 | EMT  | EMulator<br>Trap | Trap to<br>location 30.<br>This is used<br>to call system<br>programs.                          | * * * * |
| 104400-<br>104777 | TRAP | TRAP             | Trap to<br>location 34.<br>This is used<br>to call any<br>routine desired<br>by the programmer. | * * * * |

## C.5.6 Branch Instructions OP E (where $-128(\text{decimal}) < (E-. -2) / 2 < 127(\text{decimal})$ )

| Op-Code | Mnemonic         | Stands for   | Condition to be<br>met if branch is<br>to occur |
|---------|------------------|--|---|
| 0004XX  | BR               | BRanch always  |   |
| 0010XX  | BNE              | Branch if Not Equal (to zero)                        | Z=0   |
| 0014XX  | BEQ              | Branch if EQUAL (to zero)                            | Z=1   |
| 0020XX  | BGE              | Branch if Greater than or<br>Equal (to zero)         | N (I) V=0                                       |
| 0024XX  | BLT              | Branch if Less Than (zero)                           | N (I) V = 1                                     |
| 0030XX  | BGT              | Branch if Greater Than<br>(zero)                     | Z! (N (I) V)=0                                  |
| 0034XX  | BLE              | Branch if Less than or<br>Equal (to zero)            | Z! (N (I) V)=1                                  |
| 1000XX  | BPL              | Branch if PLus                                       | N=0   |
| 1004XX  | BMI              | Branch if MInus                                      | N=1   |
| 1010XX  | BHI              | Branch if HIGher                                     | C (I) Z=0                                       |
| 1014XX  | BLOS             | Branch if LOwer or Same                              | C! Z=1  |
| 1020XX  | BVC              | Branch if oVerflow Clear                             | V=0   |
| 1024XX  | BVS              | Branch if oVerflow Set                               | V=1   |
| 1030XX  | BCC (or<br>BHIS) | Branch if Carry Clear<br>(or Branch if High or Same) | C=0   |
| 1034XX  | BCS (or<br>BLO)  | Branch if Carry Set (or<br>Branch if LOw)            | C=1   |



# MACRO Assembler, Instruction, and Character Code Summaries

## C.5.7 Register Destination (OP ER,A)

| Op-Code | Mnemonic | Stands for         | Operation   | Status Word Condition Codes |   |   |   |
|---------|----------|--------------------|---|-----------------------------|---|---|---|
|         |          |                    |   | N                           | Z | V | C |
| 004RDD  | JSR      | Jump to SubRoutine | Push register on the SP stack, put the PC in the register:<br><br>DE TEMP (TEMP= temporary storage register internal to processor.)<br><br>(SP)-2 → SP<br>(REG) → (SP)<br>(PC) → REG<br>(TEMP) → PC | -                           | - | - | - |

The following instruction is available only on the 11/35, 11/40, 11/45:

|        |     |              |                     |   |   |   |   |
|--------|-----|--------------|---------------------|---|---|---|---|
| 074RDD | XOR | eXclusive OR | (R) (1) (DE) → (DE) | * | * | 0 | - |
|--------|-----|--------------|---------------------|---|---|---|---|

## C.5.8 Register-Offset (OP R,E)

The following instruction is available only on the PDP-11/35, 11/40, 11/45:

| Op-Code | Mnemonic | Stands for              | Operation                         | N | Z | V | C |
|---------|----------|-------------------------|-----------------------------------|---|---|---|---|
| 077RDD  | SOB      | Subtract One and Branch | (R)-1 → (R)<br>(PC)-(2*DE) → (PC) | - | - | - | - |

## C.5.9 Subroutine Return (OP ER)

| Op-Code | Mnemonic | Stands for             | Operation  | N | Z | V | C |
|---------|----------|------------------------|--|---|---|---|---|
| 00020R  | RTS      | ReTurn from Subroutine | Put register in PC and pop old contents from SP stack into register. | - | - | - | - |

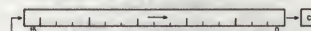


# MACRO Assembler, Instruction, and Character Code Summaries

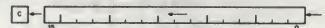
## C.5.10 Source-Register (OP A,R)

The following instructions are available on the 11/35, 11/40, 11/45 only:

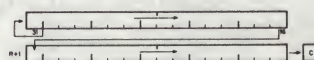
| Op-Code | Mnemonic | Stands for          | Operation   | Status Word<br>Floating<br>Condition<br>Codes |   |   |   |
|---------|----------|---------------------|---|---|---|---|---|
|         |          |                     |   | N   | Z | V | C |
| 071RSS  | DIV      | DIVide              | $(R), (R11) / (SRC) \rightarrow (R), (R11)$                   | *   | * | * | * |
| 070RSS  | MUL      | MULtiply            | $(R) * (SRC) \rightarrow (R), (R11)$                          | *   | * | * | * |
| 072RSS  | ASH      | Arithmetic<br>Shift | R is shifted<br>according to<br>low-order 6-bits<br>of source |   |   |   |   |



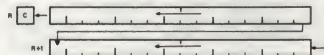
or



|        |      |                                 |   |   |   |   |   |
|--------|------|---------------------------------|---|---|---|---|---|
| 073RSS | ASHC | Arithmetic<br>Shift<br>Combined | R,RVL are shifted<br>according to low-<br>order 6 bits of<br>source | * | * | * | * |
|--------|------|---------------------------------|---|---|---|---|---|



or



## C.5.11 Floating-Point Source Double Register (OP A,AC)

The following instructions are available on the PDP-11/45 with FP11-B only:

| Op-Code   | Mnemonic | Stands for | Operation                       | Status Word<br>Floating<br>Condition<br>Codes |    |    |    |
|-----------|----------|------------|---------------------------------|---|----|----|----|
|           |          |            |                                 | FN  | FZ | FV | FC |
| 172(AC)SS | ADDD     | ADD Double | $(FSE) + (AC) \rightarrow (AC)$ | *   | *  | *  | 0  |



# MACRO Assembler, Instruction, and Character Code Summaries

|             |       |   |  |   |   |   |   |
|-------------|-------|---|--|---|---|---|---|
| 172(AC)SS   | ADDF  | ADD Floating  | (FSE) $+(AC) \rightarrow (AC)$         | * | * | * | 0 |
| 173(AC+4)SS | CMPD  | CoMPare Double                                      | (FSE) $-(AC)$                          | * | * | 0 | 0 |
| 173(AC+4)SS | CMFF  | CoMPare Floating                                    | (FSE) $-(AC)$                          | * | * | 0 | 0 |
| 174(AC+4)SS | DIVD  | DIVide Double                                       | $(AC)/(FSE) \rightarrow (AC)$          | * | * | * | 0 |
| 174(AC+4)SS | DIVF  | DIVide Floating                                     | $(AC)/(FSE) \rightarrow (AC)$          | * | * | * | 0 |
| 177(AC+4)SS | LDCDF | LoaD and Con-<br>vert from<br>Double to<br>Floating | (FSE) $\rightarrow (AC)$               | * | * | * | 0 |
| 177(AC+4)SS | LDCFD | LoaD and Con-<br>vert from<br>Floating to<br>Double | (FSE) $\rightarrow (AC)$               | * | * | * | 0 |
| 172(AC+4)SS | LDD   | LoaD Double   | (FSE) $\rightarrow (AC)$               | * | * | 0 | 0 |
| 172(AC+4)SS | LDF   | LoaD Floating                                       | (FSE) $\rightarrow (AC)$               | * | * | 0 | 0 |
| 171(AC+4)SS | MODD  | Multiply and<br>integerize<br>double                | $(AC) * (FSE) \rightarrow (AC), (AC1)$ | * | * | * | * |
| 171(AC+4)SS | MODF  | Multiply and<br>integerize<br>floating-<br>point    | $(AC) * (FSE) \rightarrow (AC)$        | * | * | * | 0 |
| 171(AC)SS   | MULD  | MULTiply Double                                     | $(AC) * (FSE) \rightarrow (AC)$        | * | * | * | 0 |
| 171(AC)SS   | MULF  | MULTiply Floating                                   | $(AC) * (FSE) \rightarrow (AC)$        | * | * | * | 0 |
| 173(AC)SS   | SUBD  | SUBtract Double                                     | (FSE) $-(AC) \rightarrow (AC)$         | * | * | * | 0 |
| 173(AC)SS   | SUBF  | SUBtract Floating                                   | (FSE) $-(AC) \rightarrow (AC)$         | * | * | * | 0 |



# MACRO Assembler, Instruction, and Character Code Summaries

## C.5.12 Source-Double Register (OP A,AC)

The following instructions are available on the PDP-11/45 with FP11-B only:

| Op-Code     | Mnemonic | Stands for                                | Operation           | Status Word<br>Condition Codes |    |    |    |
|-------------|----------|---|---------------------|--------------------------------|----|----|----|
|             |          |   |                     | FN                             | FZ | FV | FC |
| 177(AC)SS   | LDCID    | Load and Convert Integer to Double        | (SE) → (AC)         | *                              | *  | *  | 0  |
| 177(AC)SS   | LDCIF    | Load and Convert Integer to Floating      | (SE) → (AC)         | *                              | *  | *  | 0  |
| 177(AC)SS   | LDCLD    | Load and Convert Long integer to Double   | (SE) → (AC)         | *                              | *  | *  | 0  |
| 177(AC)SS   | LDCLF    | Load and Convert Long Integer to Floating | (SE) → (AC)         | *                              | *  | *  | 0  |
| 176(AC+4)SS | LDEXP    | Load EXPonent                             | (SE)+200 → (AC EXP) | *                              | *  | 0  | 0  |

## C.5.13 Double Register-Destination (OP AC,A)

The following instructions are available on the PDP-11/45 with FP11-B only:

| Op-Code     | Mnemonic | Stands for                             | Operation    | Status Word<br>Condition Codes |    |    |    |
|-------------|----------|--|--------------|--------------------------------|----|----|----|
|             |          |  |              | FN                             | FZ | FV | FC |
| 176(AC)DD   | STCFD    | STore, Convert from Floating to Double | (AC) → (FDE) | *                              | *  | *  | 0  |
| 176(AC)DD   | STCDF    | STore, Convert from Double to Floating | (AC) → (FDE) | *                              | *  | *  | 0  |
| 175(AC+4)DD | STCDI(1) | STore, Convert from Double to Integer  | (AC) → (FDE) | *                              | *  | 0  | *  |

(1) These instructions set both the floating-point and processor condition codes as indicated.



## MACRO Assembler, Instruction, and Character Code Summaries

|                     |          |   |                     |   |   |   |   |
|---------------------|----------|---|---------------------|---|---|---|---|
| 175(AC+4) DD        | STCDL(1) | STore, Con-<br>vert from<br>Double to<br>Long integer   | (AC) → (FDE)        | * | * | 0 | * |
| 175(AC+4) DD        | STCFI(1) | STore, Con-<br>vert from<br>Floating to<br>Integer      | (AC) → (FDE)        | * | * | 0 | * |
| 174(AC+4) DD        | STCFL(1) | STore, Con-<br>vert from<br>Floating to<br>Long integer | (AC) → (FDE)        | * | * | 0 | * |
| 174(AC) DD STD      |          | STore Double  | (AC) → (FDE)        | - | - | - | - |
| 174(AC) DD STF      |          | STore<br>Floating                                       | (AC) → (FDE)        | - | - | - | - |
| 175(AC) DD STEXP(1) |          | STore<br>EXPonent                                       | (AC EXP)-200 → (DE) | * | * | 0 | 0 |

### C.5.14 Number

The following instruction is available on the 11/35, 11/40, 11/45 only:

| Op-Code | Mnemonic | Stands for | Operation  | Status Word<br>Condition Codes |   |   |   |
|---------|----------|------------|--|--------------------------------|---|---|---|
|         |          |            |  | N                              | Z | V | C |
| 0064NN  | MARK     | MARK       | Stack cleanup on<br>return from sub-<br>routine. | -                              | - | - | - |

### C.5.15 Priority

The following instruction is available on the PDP-11/45 only:

| Op-Code | Mnemonic | Stands for            | Operations               | Status Word<br>Condition<br>Codes |   |   |   |
|---------|----------|-----------------------|--------------------------|-----------------------------------|---|---|---|
|         |          |                       |                          | N                                 | Z | V | C |
| 00023N  | SPL      | Set Priority<br>Level | (X) → (PS)<br>(bits 7-5) | -                                 | - | - | - |

(1) These instructions set both the floating-point and processor condition codes as indicated.



# MACRO Assembler, Instruction, and Character Code Summaries

## C.6 ASSEMBLER DIRECTIVES

| <u>Form</u>   | <u>Described in<br/>Manual Section</u> | <u>Operation</u>   |
|---------------|--|--|
| '             | 5.5.3.3                                | A single quote character (apostrophe) followed by one ASCII character generates a word containing the 7-bit ASCII representation of the character in the low-order byte and zero in the high-order byte. |
| "             | 5.5.3.3                                | A double quote character followed by two ASCII characters generates a word containing the 7-bit ASCII representation of the two characters.  |
| \             | 5.6.3.3                                | A backslash preceding an argument causes the number to be treated in the current radix.  |
| †Bn           | 5.5.4.2                                | Temporary radix control; causes the number n to be treated as a binary number.   |
| †Cn           | 5.5.6.2                                | Creates a word containing the one's complement of n.   |
| †Dn           | 5.5.4.2                                | Temporary radix control; causes the number n to be treated as a decimal number.  |
| †Fn           | 5.5.6.2                                | Creates a one-word floating point quantity to represent n.   |
| †On           | 5.5.4.2                                | Temporary radix control; causes the number n to be treated as an octal number.   |
| .ASCII string | 5.5.3.4                                | Generates a block of data containing the ASCII equivalent of the character string (enclosed in delimiting characters) one character per byte.  |
| .ASCIZ string | 5.5.3.5                                | Generates a block of data containing the ASCII equivalent of the character string (enclosed in delimiting characters) one character per byte with a zero byte following the specified string.            |
| .ASECT        | 5.5.9                                  | Begin or resume absolute section.  |
| .BLKB exp     | 5.5.5.3                                | Reserves a block of storage space exp bytes long.  |
| .BLKW exp     | 5.5.5.3                                | Reserves a block of storage space exp words long.  |



## MACRO Assembler, Instruction, and Character Code Summaries

|                          |         |  |
|--------------------------|---------|--|
| .BYTE expl,<br>exp2,...  | 5.5.3.1 | Generates successive bytes of data containing the octal equivalent of the expression(s) specified.   |
| .CSECT symbol            | 5.5.9   | Begins or resumes named or unnamed relocatable section.  |
| .DSABL arg               | 5.5.2   | Disables the assembler function specified by the argument.   |
| .ENABL arg               | 5.5.2   | Provides the assembler function specified by the argument.   |
| .END<br>.END exp         | 5.5.7.1 | Indicates the physical end of the source program. An optional argument specifies the transfer address.   |
| .ENDC                    | 5.5.11  | Indicates the end of a condition block.  |
| .ENDM<br>.ENDM symbol    | 5.6.1.2 | Indicates the end of the current repeat block, indefinite repeat block, or macro. The optional symbol, if used, must be identical to the macro name.                             |
| .EOT                     | 5.5.7.2 | Ignored. Indicates End-of-Tape which is detected automatically by the hardware.  |
| .ERROR exp,string        | 5.6.5   | Causes a text string to be output to the listing containing the optional expression specified and the indicated text string. The line will be flagged with the "P" error code.   |
| .EVEN                    | 5.5.5.1 | Ensures that the assembly location counter contains an even address by adding 1 if it is odd.  |
| .FLT2 arg1,<br>arg2,...  | 5.5.6.1 | Generates successive two-word floating point equivalents for the floating-point numbers specified as arguments.  |
| .FLT4 arg1,<br>arg2,...  | 5.5.6.1 | Generates successive four-word floating point equivalents for the floating-point numbers specified as arguments.   |
| .GLOBL sym1,<br>sym2,... | 5.5.10  | Defines the symbol(s) specified as global symbol(s).   |
| .IDENT symbol            | 5.5.1.5 | Provides a means of labeling the object module produced as a result of assembly. This directive is not supported by RT-11, but is included for compatibility with other systems. |
| .IF cond,<br>arguments   | 5.5.11  | Begins a conditional block of source code which is included in the assembly  |



## MACRO Assembler, Instruction, and Character Code Summaries

|                              |          |  |
|------------------------------|----------|--|
|                              |          | only if the stated condition is met with respect to the argument(s) specified.   |
| .IFF                         | 5.5.11.1 | Appears only within a conditional block and indicates the beginning of a section of code to be assembled if the condition tested false.  |
| .IFT                         | 5.5.11.1 | Appears only within a conditional block and indicates the beginning of a section of code to be assembled if the condition tested true.   |
| .IFTF                        | 5.5.11.1 | Appears only within a conditional block and indicates the beginning of a section of code to be unconditionally assembled.  |
| .IIF cond,arg,<br>statement  | 5.5.11.2 | Acts as a one-line conditional block where the condition is tested for the argument specified. The statement is assembled only if the condition tests true.  |
| .IRP sym,<br><arg1,arg2,...> | 5.6.6    | Indicates the beginning of an indefinite repeat block in which the symbol specified is replaced with successive elements of the real argument list (which is enclosed in angle brackets).                |
| .IRPC sym,string             | 5.6.6    | Indicates the beginning of an indefinite repeat block in which the symbol specified takes on the value of successive characters in the character string.   |
| .LIMIT                       | 5.5.8    | Reserves two words into which the Linker inserts the low and high addresses of the relocated code.   |
| .LIST<br>.LIST arg           | 5.5.1.1  | Without an argument, .LIST increments the listing level count by 1. With an argument, .LIST does not alter the listing level count but formats the assembly listing according to the argument specified. |
| .MACRO sym,arg1,<br>arg1,... | 5.6.1.1  | Indicates the start of a macro with the specified name containing the dummy arguments specified.   |
| .MCALL                       | 5.6.8    | Used to specify the names of all system macro definitions not defined in the current program but required by the program.  |
| .MEXIT                       | 5.6.1.3  | Causes an exit from the current macro or indefinite repeat block.  |



## MACRO Assembler, Instruction, and Character Code Summaries

|  |         |  |
|--|---------|--|
| <code>.NARG symbol</code>                      | 5.6.4   | Appears only within a macro definition and equates the specified symbol to the number of arguments in the macro call currently being expanded.   |
| <code>.NCHR sym,&lt;string&gt;</code>          | 5.6.4   | Can appear anywhere in a source program; equates the symbol specified to the number of characters in the string (enclosed in delimiting characters).   |
| <code>.NLIST</code><br><code>.NLIST arg</code> | 5.5.1.1 | Without an argument, <code>.NLIST</code> decrements the listing level count by 1. With an argument, <code>.NLIST</code> deletes the portion of the listing indicated by the argument.                              |
| <code>.NTYPE symbol,arg</code>                 | 5.6.4   | Appears only in a macro definition and equates the low-order six bits of the symbol specified to the six-bit addressing mode of the argument.  |
| <code>.ODD</code>                              | 5.5.5.2 | Ensures that the assembly location counter contains an odd address by adding 1 if it is even.  |
| <code>.PAGE</code>                             | 5.5.1.6 | Causes the assembly listing to skip to the top of the next page.   |
| <code>.PRINT exp,string</code>                 | 5.6.5   | Causes a text string to be output to the listing containing the optional expression specified and the indicated text string.   |
| <code>.RADIX n</code>                          | 5.5.4.1 | Alters the current program radix to n, where n can be 2, 4, 8, or 10.  |
| <code>.RAD50 string</code>                     | 5.5.3.6 | Generates a block of data containing the Radix-50 equivalent of the character string (enclosed in delimiting characters).  |
| <code>.REPT exp</code>                         | 5.6.7   | Begins a repeat block. Causes the section of code up to the next <code>.ENDM</code> or <code>.ENDR</code> to be repeated exp times.  |
| <code>.SBTTL string</code>                     | 5.5.1.4 | Causes the string to be printed as part of the assembly listing page header. The string part of each <code>.SBTTL</code> directive is collected into a table of contents at the beginning of the assembly listing. |
| <code>.TITLE string</code>                     | 5.5.1.3 | Assigns the first symbolic name in the string to the object module and causes the string to appear on each page of the assembly listing. One <code>.TITLE</code> directive should be issued per program.           |



## MACRO Assembler, Instruction, and Character Code Summaries

|                         |         |  |
|-------------------------|---------|--|
| .WORD expl,<br>exp2,... | 5.5.3.2 | Generates successive words of data containing the octal equivalent of the expression(s) specified. |
|-------------------------|---------|--|

### C.7 MACRO/CREF SWITCHES

#### C.7.1 Listing Control Switches

| <u>Switch</u>    | <u>Meaning</u>  |
|------------------|---|
| /L:arg<br>/N:arg | These switches are used to control listing output.<br>Arguments which are valid for either switch include:  |
| <u>Arg</u>       | <u>Controls Listing of:</u>   |
| SEQ              | Source line sequence numbers  |
| LOC              | Location counter  |
| BIN              | Generated binary code   |
| BEX              | Binary extensions   |
| SRC              | Source code   |
| COM              | Comments  |
| MD               | Macro definitions and repeat range definitions  |
| MC               | Macro calls and repeat range expansions   |
| ME               | Macro expansions  |
| MEB              | Macro expansion binary code   |
| CND              | Unsatisfied conditions and all .IF and .ENDC statements.  |
| LD               | Listing directives having no arguments  |
| TOC              | Table of contents   |
| TTM              | Listing output format   |
| SYM              | Symbol table  |
| <no arg>         | /N with no argument causes only table of contents, symbol table, and error listings to be produced.<br><br>/L with no argument causes .LIST and .NLIST directives without arguments which appear in the source program to be ignored. |

#### C.7.2 Function Control Switches

| <u>Switch</u>    | <u>Meaning</u>   |
|------------------|--|
| /D:arg<br>/E:arg | These switches are used to enable or disable certain functions in source input files. Valid arguments include: |
| <u>Arg</u>       | <u>Enables or Disables:</u>  |
| ABS              | Absolute binary output   |
| AMA              | Assembly of all absolute addresses as relative addresses   |
| CDR              | Source columns 73 and greater to be treated as comments  |
| FPT              | Floating point truncation  |
| LC               | Accepts lower case ASCII input   |
| LSB              | Local symbol block   |
| PNC              | Binary output  |



## MACRO Assembler, Instruction, and Character Code Summaries

### C.7.3 CREF Switches

| <u>Switch</u> | <u>Produces Cross-Reference of:</u> |
|---------------|-------------------------------------|
| /C:S          | User-defined symbols                |
| /C:R          | Register symbols                    |
| /C:M          | MACRO symbolic names                |
| /C:P          | Permanent symbols                   |
| /C:C          | Control sections                    |
| /C:E          | Error codes                         |
| /C:<no arg>   | Equivalent to /C:S:M:E              |



# APPENDIX D

## SYSTEM MACRO FILE

The following is a listing of the system macro library, SYSMAC.SML. This file is stored on the system device, and used by MACRO when it expands the programmed requests discussed in Chapter 9.

```

*****
;   DEC ASSUMES NO RESPONSIBILITY FOR
;   THE USE OR RELIABILITY OF ITS
;   SOFTWARE ON EQUIPMENT WHICH IS NOT
;   SUPPLIED BY DEC.
*****
;   SYSMAC.SML--SYSTEM MACRO LIBRARY
;   FOR RT11 V2.
;
;   DEC-11-ORSYA-A-LA
;
;   COPYRIGHT 1974
;   DIGITAL EQUIPMENT CORPORATION
;   MAYNARD, MASSACHUSETTS 01754
;   EF
;
*****

```

```

.MACRO ..V2..
.MCALL ...CM1,...CM2,...CM3,...CM4
...V2=1
.ENDM

```

```

.MACRO ...CM1 ,AREA,.CODE,.CHAN
.IF NB .AREA
    MOV    .AREA,X0
    MOV    #.CODE*^0400,(0)
.ENDC
.IIF NB <.CHAN>,
    MOV    .CHAN,(0)
.ENDM

```



# System Macro File

```
.MACRO ...CM2 .ARG,.OFFSET,.INS
.IIF NB <,.ARG>,      MOV      .ARG,.OFFSET(0)
.IIF NB <,.INS>,      EMT      A0375
.ENDM
```

```
.MACRO ...CM3 .CHAN,.CODE
.IIF NB <,.CHAN>,      MOV      *.CODE+A0400,X0
                      BISH      .CHAN,X0
                      EMT      A0374
.ENDM
```

```
.MACRO ...CM4 .AREA,.CHAN,.BUFF,.WCNT,.BLK,.CRTN,.CODE
...CM1 <,.AREA>,<,.CODE>,<,.CHAN>
...CM2 <,.BLK>,2.
...CM2 <,.BUFF>,4.
...CM2 <,.WCNT>,6.
...CM2 <,.CRTN>,8.,X
.ENDM
```

```
.MACRO .CUFN .AREA,.ADD,.NUM
...CM1 <,.AREA>,13.
...CM2 <,.ADD>,2.
...CM2 <,.NUM>,4.,X
.ENDM
```

```
.MACRO .CHAIN
...CM3 ,8.
.ENDM
```

```
.MACRO .CHCOPY .AREA,.CHAN,.OCHAN
...CM1 <,.AREA>,11.,<,.CHAN>
...CM2 <,.OCHAN>,2.,X
.ENDM
```

```
.MACRO .CNTXSW .AREA,.ADD
...CM1 <,.AREA>,27.
...CM2 <,.ADD>,2.,X
.ENDM
```

```
.MACRO .CMKT .AREA,.ID,.TIME
...CM1 <,.AREA>,19.
...CM2 <,.ID>,2.
.IF B .TIME
```

```
                      CLR      4,(0)
.IFF
                      MOV      .TIME,4.(0)
.ENDC
                      EMT      A0375
.ENDM
```

```
.MACRO .CLOSE .CHAN
.IF NDF ...V2
                      EMT      A0<100+.CHAN>
.IFF
...CM3 <,.CHAN>,6.
.ENDC
.ENDM
```



# System Macro File

```
.MACRO .CSIGEN .DEVSPC,.DEFEXT,.CSTRING
    MOV    .DEVSPC,=(6.)
    MOV    .DEFEXT,=(6.)
    IF B .CSTRING
        CLR    =(6.)
    IFF
        MOV    .CSTRING,=(6.)
    ENDC
    EMT      ^0344
.ENDM
```

```
.MACRO .CSISPC .OUTSPC,.DEFEXT,.CSTRING
    MOV    .OUTSPC,=(6.)
    MOV    .DEFEXT,=(6.)
    IF B .CSTRING
        CLR    =(6.)
    IFF
        MOV    .CSTRING,=(6.)
    ENDC
    EMT      ^0345
.ENDM
```

```
.MACRO .CSTAT .AREA,.CHAN,.ADD
...CM1 <.AREA>,23,<.CHAN>
...CM2 <.ADD>,2,,X
.ENDM
```

```
.MACRO .DATE
    MOV    #54,X0
    MOV    ^0262(0),X0
.ENDM
```

```
.MACRO .DELETE .AREA,.CHAN,.DEVBLK,.SPF
    IF NDF ...V2
    IIF NB <.CHAN>
        MOV    .CHAN,X0
        EMT      ^0<.AREA>
    IFF
        ...CM1 <.AREA>,0,<.CHAN>
        ...CM2 <.DEVBLK>,2.
        IF B .SPF
            CLR    4.(0)
        IFF
            MOV    .SPF,4.(0)
        ENDC
        EMT      ^0375
    ENDC
.ENDM
```

```
.MACRO .DEVICE .AREA,.ADD
...CM1 <.AREA>,12.
...CM2 <.ADD>,2,,X
.ENDM
```

```
.MACRO .DSTATUS .RETSPC,.DNAME
    IIF NB <.DNAME>,
        MOV    .DNAME,X0
        MOV    .RETSPC,=(6.)
        EMT      ^0342
.ENDM
```



# System Macro File

```

.MACRO .ENTER .AREA,.CHAN,.DEVBLK,.LEN,.SPF
.IF NUF ...V2
    MOV     .CHAN,X0
    .IF B .DEVBLK
        CLR     =(6.)
    .IFF
        MOV     .DEVBLK,=(6.)
    .ENDC
        EMT     A0<40+.AREA>
    .IFF
        ...CM1 <.AREA>,2.,<.CHAN>
        ...CM2 <.DEVBLK>,2.
        .IF NB .LEN
            MOV     .LEN,4.(0)
        .IFF
            CLR     4.(0)
        .ENDC
        .IF NB .SPF
            MOV     .SPF,6.(0)
        .IFF
            CLR     6.(0)
        .ENDC
        EMT     A0375
    .ENDC
.ENDM

.MACRO .EXIT
        EMT     A0350
.ENDM

.MACRO .FETCH .ADD,.DNAME
.IF NB <.DNAME>,
    MOV     .DNAME,X0
    MOV     .ADD,=(6.)
    EMT     A0343
.ENDM

.MACRO .GTIM .AREA,.ADD
...CM1 <.AREA>,17.
...CM2 <.ADD>,2.,X
.ENDM

.MACRO .GTJB .AREA,.ADD
...CM1 <.AREA>,16.
...CM2 <.ADD>,2.,X
.ENDM

.MACRO .HERR
...CM3 ,5.
.ENDM

.MACRO .HRESET
        EMT     A0357
.ENDM

```



System Macro File

```

.MACRO .INTEN ,PRIO,.PIC
.IF NB .PIC
    MOV    #A054,-(6.)
    JSR    5.,0(6,)+
.IFF
    JSR    5.,0A054
.ENDC
    .WORD  AC<,PRIO*32.>8224.
.ENDM

.MACRO .LOCK
    EMT    A0346
.ENDM

.MACRO .LOOKUP .AREA,.CHAN,.DEVBLK,.SPF
.IF NDF ...V2
    .IF NB <,.CHAN>,
        MOV    ,CHAN,X0
        EMT    A0<20+,.AREA>
    .IFF
    ...CM1 <,.AREA>,1,<,.CHAN>
    ...CM2 <,.DEVBLK>,2.
    .IF B .SPF
        CLR    4.(0)
    .IFF
        MOV    .SPF,4.(0)
    .ENDC
    EMT    A0375
.ENDC
.ENDM

.MACRO .MRKT .AREA,.TIME,.CRTN,.ID
...CM1 <,.AREA>,18.
...CM2 <,.TIME>,2.
...CM2 <,.CRTN>,4.
...CM2 <,.ID>,6.,X
.ENDM

.MACRO .MWAIT
...CM3 ,9.
.ENDM

.MACRO .PRINT .ADD
.IF NB <,.ADD>,
    MOV    .ADD,X0
    EMT    A0351
.ENDM

.MACRO .PROTECT .AREA,.ADD
...CM1 <,.AREA>,25.
...CM2 <,.ADD>,2.,X
.ENDM

.MACRO .PURGE .CHAN
...CM3 <,.CHAN>,3.
.ENDM

.MACRO .QSET .QADD,.QLEN
.IF NB <,.QLEN>,
    MOV8   .QLEN,X0
    MOV    .QADD,-(6.)
    EMT    A0353
.ENDM

```



System Macro File

```

.MACRO .RCTRLO
                                EMT      A0355
.ENDM

.MACRO .RCVD .AREA,.BUFF,.WCNT
...CM4 <.AREA>,,<.BUFF>,<.WCNT>,,#1,22.
.ENDM

.MACRO .RCVDC .AREA,.BUFF,.WCNT,.CRTN
...CM4 <.AREA>,,<.BUFF>,<.WCNT>,,<.CRTN>,22.
.ENDM

.MACRO .RCVDW .AREA,.BUFF,.WCNT
...CM4 <.AREA>,,<.BUFF>,<.WCNT>,,#0,22.
.ENDM

.MACRO .READ .AREA,.CHAN,.BUFF,.WCNT,.BLK
.IF NDF ...V2
.IIF NB <.WCNT>,
                                MOV      .WCNT,X0
                                MOV      #1,-(6.)
                                MOV      .BUFF,-(6.)
                                MOV      .CHAN,-(6.)
                                EMT      A0<200+,.AREA>
.IFF
...CM4 <.AREA>,<.CHAN>,<.BUFF>,<.WCNT>,<.BLK>,#1,8.
.ENDC
.ENDM

.MACRO .READC .AREA,.CHAN,.BUFF,.WCNT,.CRTN,.BLK
.IF NDF ...V2
.IIF NB <.CRTN>,
                                MOV      .CRTN,X0
                                MOV      .WCNT,-(6.)
                                MOV      .BUFF,-(6.)
                                MOV      .CHAN,-(6.)
                                EMT      A0<200+,.AREA>
.IFF
...CM4 <.AREA>,<.CHAN>,<.BUFF>,<.WCNT>,<.BLK>,<.CRTN>,8.
.ENDC
.ENDM

.MACRO .READW .AREA,.CHAN,.BUFF,.WCNT,.BLK
.IF NDF ...V2
.IIF NB <.WCNT>,
                                MOV      .WCNT,X0
                                CLR      -(6.)
                                MOV      .BUFF,-(6.)
                                MOV      .CHAN,-(6.)
                                EMT      A0<200+,.AREA>
.IFF
...CM4 <.AREA>,<.CHAN>,<.BUFF>,<.WCNT>,<.BLK>,#0,8.
.ENDC
.ENDM

.MACRO .REGDEF
R0=X0
R1=X1
R2=X2
R3=X3
R4=X4
R5=X5
SP=X6
PC=X7
.ENDM

```



System Macro File

```

.MACRO .RELEASE .DEVBLK
.IIF NB <.DEVBLK>,      MOV    .DEVBLK,X0
                        CLR    =(6,)
                        EMT     A0343

.ENDM

.MACRO .RENAME .AREA,.CHAN,.DEVBLK
.IF NDF ...V2
.IIF NB <.CHAN>,      MOV    .CHAN,X0
                        EMT     A0<100+.AREA>

.IFF
...CM1 <.AREA>,4,,<.CHAN>
...CM2 <.DEVBLK>,2,,X
.ENDC
.ENDM

.MACRO .REOPEN .AREA,.CHAN,.CBLK
.IF NDF ...V2
.IIF NB <.CHAN>,      MOV    .CHAN,X0
                        EMT     A0<140+.AREA>

.IFF
...CM1 <.AREA>,6,,<.CHAN>
...CM2 <.CBLK>,2,,X
.ENDC
.ENDM

.MACRO .SAVESTAT .AREA,.CHAN,.CBLK
.IF NDF ...V2
.IIF NB <.CHAN>,      MOV    .CHAN,X0
                        EMT     A0<120+.AREA>

.IFF
...CM1 <.AREA>,5,,<.CHAN>
...CM2 <.CBLK>,2,,X
.ENDC
.ENDM

.MACRO .RSUM

...CM3 ,2.
.ENDM

.MACRO .SDAT .AREA,.BUFF,.WCNT
...CM4 <.AREA>,,<.BUFF>,<.WCNT>,,#1,21.
.ENDM

.MACRO .SUATC .AREA,.BUFF,.WCNT,.CRTN
...CM4 <.AREA>,,<.BUFF>,<.WCNT>,,<.CRTN>,21.
.ENDM

.MACRO .SDATW .AREA,.BUFF,.WCNT
...CM4 <.AREA>,,<.BUFF>,<.WCNT>,,#0,21.
.ENDM

.MACRO .SERK
...CM3 ,4.
.ENDM

```



# System Macro File

```

.MACRO .SETTOP .ADD
.IIF NB < .ADU>,          MOV      .ADD,%0
                          EMT      ^0354
.ENDM

.MACRO .SFPA .AREA,.ADD
...CM1 < .AREA>,24.
...CM2 < .ADU>,2.,X
.ENDM

.MACRO .SPFUN .AREA,.CHAN,.CODE,.BUFF,.WCNT,.BLK,.CRTN
...CM1 < .AREA>,26.,< .CHAN>
...CM2 < .BLK>,2.
...CM2 < .BUFF>,4.
...CM2 < .WCNT>,6.
.IF NB .CODE
                          MOV8     #^0377,8.(0)
                          MOV8     .CODE,9.(0)
.ENDC
.IF B .CRTN
                          CLR      10.(0)
.IFF
                          MOV      .CRTN,10.(0)
.ENDC
                          EMT      ^0375
.ENDM

.MACRO .SRESET
                          EMT      ^0352
.ENDM

.MACRO .SPND
...CM3 ,1
.ENDM

.MACRO .SYNCH .AREA
.IIF NB < .AREA>,          MOV      .AREA,%4
                          MOV      #^054,%5
                          JSR      5.,0^0324(5.)
.ENDM

.MACRO .TLOCK
...CM3 ,7.
.ENDM

.MACRO .TRPSET .AREA,.ADD
...CM1 < .AREA>,3.
...CM2 < .ADU>,2.,X
.ENDM

.MACRO .TTINR
                          EMT      ^0340
.ENDM

.MACRO .TIYIN .CHAR
                          EMT      ^0340
                          BCS      ,=2
                          MOV8     %0,.CHAR
.IIF NB < .CHAR>,
.ENDM

```



System Macro File

```

.MACRO .TTOUTR
                                EMT      A0341
.ENDM

.MACRO .TTYOUT .CHAR
.IF NB < .CHAR >,              MOV      .CHAR, X0
                                EMT      A0341
                                DCS      .-2
.ENDM

.MACRO .TWAIT .AREA, .TIME
...CM1 < .AREA >, 20,
...CM2 < .TIME >, 2,, X
.ENDM

.MACRO .UNLOCK
                                EMT      A0347
.ENDM

.MACRO .WAIT .CHAN
.IF NDF ...V2
                                EMT      A0<240+ .CHAN>
.IFF
...CM3 < .CHAN >, 0
.ENDC
.ENDM

.MACRO .WRITE .AREA, .CHAN, .BUFF, .WCNT, .BLK
.IF NDF ...V2
.IIF NB < .WCNT >,             MOV      .WCNT, X0
                                MOV      #1, -(6.)
                                MOV      .BUFF, -(6.)
                                MOV      .CHAN, -(6.)
                                EMT      A0<220+ .AREA>
.IFF
...CM4 < .AREA >, < .CHAN >, < .BUFF >, < .WCNT >, < .BLK >, #1, 9.
.ENDC
.ENDM

.MACRO .WRITW .AREA, .CHAN, .BUFF, .WCNT, .BLK
.IF NDF ...V2
.IIF NB < .WCNT >,             MOV      .WCNT, X0
                                CLR      -(6.)
                                MOV      .BUFF, -(6.)
                                MOV      .CHAN, -(6.)
                                EMT      A0<220+ .AREA>
.IFF

...CM4 < .AREA >, < .CHAN >, < .BUFF >, < .WCNT >, < .BLK >, #0, 9.
.ENDC
.ENDM

.MACRO .WRITC .AREA, .CHAN, .BUFF, .WCNT, .CRTN, .BLK
.IF NDF ...V2
.IIF NB < .CRTN >,             MOV      .CRTN, X0
                                MOV      .WCNT, -(6.)
                                MOV      .BUFF, -(6.)
                                MOV      .CHAN, -(6.)
                                EMT      A0<220+ .AREA>
.IFF
...CM4 < .AREA >, < .CHAN >, < .BUFF >, < .WCNT >, < .BLK >, < .CRTN >, 9.
.ENDC
.ENDM

```



Page 108

1900

1901

1902

1903

1904

1905

1906

1907

1908

1909

1910

1911

1912

1913

1914

1915

1916

1917

1918

1919

1920

1921

1922

1923

1924

1925

1926

1927

1928

1929

1930

1931

1932

1933

1934

1935

1936

1937

1938

1939

1940



APPENDIX E  
PROGRAMMED REQUEST SUMMARY

E.1 PARAMETERS

The following parameters are used as arguments in various calls. (Any parameters used which are not mentioned here are particular to a request and the appropriate section in Chapter 9 should be consulted.)

| <u>Parameter</u> | <u>Description</u>  |
|------------------|---|
| .addr            | an address, the meaning of which depends on the request being used  |
| .area            | a pointer to the EMT argument list  |
| .blk             | a block number specifying the relative block in a file where an I/O operation is to begin                   |
| .buff            | a buffer address specifying a memory location into which or from which an I/O transfer is to be performed   |
| .chan            | a channel number in the range 0-377 (octal)   |
| .crtn            | the entry point of a completion routine   |
| .count           | file number for magtape/cassette operations   |
| .dblkl           | the address of a four-word RAD50 descriptor of the file to be opened  |
| .num             | a number, the value of which depends on the request   |
| .wcnt            | a word count specifying the number of words to be transferred to or from the buffer during an I/O operation |

E.2 REQUEST SUMMARY

Refer to Appendix D (SYSMAC.SML) to see how each macro call is expanded in assembly language code.



# Programmed Request Summary

| <u>Mnemonic</u> | <u>Function</u>   | <u>Macro Call</u>          | <u>Error Codes<br/>(Byte 52=)</u>   |
|-----------------|---|----------------------------|---|
| .CDFN           | Increases number of I/O channels to as many as 256(decimal)   | .CDFN .area,.addr,.num     | 0 - attempt to define fewer channels than already exist   |
| .CHAIN          | Allows one back-ground program to transfer control to another without operator intervention                         | .CHAIN                     | Can produce any errors which the monitor RUN command can produce  |
| .CHCOPY         | Opens a channel for input, logically connecting it to another job open for either input or output (F/B only)        | .CHCOPY .area,.chan,.ochan | 0 - other job does not exist, or does not have enough channels defined, or does not have specified channel open |
| .CLOSE          | Terminates activity on specified channel and frees it for use in another operation; makes tentative files permanent | .CLOSE .chan               | 1 - channel already open  |
| .CMKT           | Cancels one or more outstanding mark time requests (F/B only)   | .CMKT .area,.id,.time      | Fatal monitor error if device handler is not in memory  |
|                 |   |                            | 0 - id is not 0 and mark time with that identification number not found   |



# Programmed Request Summary

| Mnemonic       | Function  | Macro Call                              | Error Codes<br>(Byte 52=)   |
|----------------|---|---|---|
| <u>.CNTXSW</u> | Specifies locations to be included in the context switch (F/B only)                               | <u>.CNTXSW .area,.addr</u>              | 0 - list is in an area into which the the USR swaps; or list is modified while the job is running.              |
| <u>.CSIGEN</u> | Calls the CSI in general mode<br><br>Note: if input is taken from TT:, all errors are printed out | <u>.CSIGEN .devspc,.defext,.cstring</u> | 0 - illegal command<br><br>1 - device not found<br>2 - unused<br>3 - full directory<br>4 - input file not found |
| <u>.CSISPC</u> | Calls the CSI in special mode<br><br>Note: if input is taken from TT:, all errors are printed out | <u>.CSISPC .outspc,.defext,.cstring</u> | 0 - illegal command line<br>1 - illegal device  |
| <u>.CSTAT</u>  | Furnishes information about a channel (F/B only)  | <u>.CSTAT .area,.chan,.addr</u>         | 0 - channel not open  |



# Programmed Request Summary

| <u>Mnemonic</u> | <u>Function</u>  | <u>Macro Call</u>                       | <u>Error Codes<br/>(Byte 52=)</u>  |
|-----------------|--|---|--|
| .DATE           | Moves current date information into R0   | .DATE                                   | None   |
| .DELETE         | Deletes named file from indicated device   | .DELETE .area,.chan,.dblk,.count        | 0 - active channel<br>1 - file not found   |
| .DEVICE         | Sets up list of addresses to be loaded with specified values upon program termination (F/B only) | .DEVICE .area,.addr                     | None   |
| .DSTATUS        | Provides information about a device  | .DSTATUS .cbk,.devnam                   | 0 - device not found   |
| .ENTER          | Allocates space on specified device and creates tentative entry for named file                   | .ENTER .area,.chan,.dblk,.length,.count | 0 - channel in use<br>1 - no space greater than or equal to the specified length was found |
| .EXIT           | Terminates user program and returns control to monitor   | .EXIT                                   | None   |
| .FETCH          | Loads device handlers into memory from system device   | .FETCH .coradd,.devnam                  | 0 - nonexistent device name or no handler for that device                                  |



# Programmed Request Summary

| <u>Mnemonic</u> | <u>Function</u>   | <u>Macro Call</u> | <u>Error Codes<br/>(Byte 52=)</u>   |
|-----------------|---|-------------------|---|
| .GTIM           | Allows access to<br>the current time of day   | .GTIM .area,.addr | None  |
| .GTJB           | Passes certain job<br>parameters back to<br>user program  | .GTJB .area,.addr | None  |
| .HERR           | Disables error<br>interception and<br>allows system to<br>detect and act<br>on normally<br>fatal errors | .HERR             | Monitor Error occurs if:<br><br>1. called USR<br>from completion<br>routine<br><br>2. no device<br>handler<br><br>3. error doing<br>directory I/O<br><br>4. FETCH error<br><br>5. Error<br>reading overlay<br><br>6. no room in<br>directory<br><br>7. illegal<br>address<br><br>10. illegal<br>channel number<br><br>11. illegal EMT |



# Programmed Request Summary

| <u>Mnemonic</u> | <u>Function</u>  | <u>Macro Call</u>                 | <u>Error Codes<br/>(Byte 52=)</u>                 |
|-----------------|--|-----------------------------------|---|
| .HRESET         | Resets channels,<br>releases device<br>handlers and stops<br>all I/O transfers<br>in progress          | .HRESET                           | None  |
| .INTEN          | Notifies monitor<br>that interrupt has<br>occurred, and sets<br>processor priority<br>to correct state | .INTEN .priority, .pic            | None  |
| .LOCK           | Locks the USR<br>in memory   | .LOCK                             | None  |
| .LOOKUP         | Associates a specified<br>channel with a<br>device and/or file<br>on that device                       | .LOOKUP .area,.chan,.dblkc,.count | 0 - channel<br>already open<br>1 - file not found |
| .MRKT           | Schedules a completion<br>routine to be<br>entered after specified<br>time interval                    | .MRKT .area,.time,.crtnc,.id      | 0 - no queue<br>element is<br>available           |
| .MWAIT          | Suspends execution<br>until all messages have<br>been transmitted or<br>received                       | .MWAIT                            | None  |
| .PRINT          | Outputs a string<br>to the terminal  | .PRINT .addr                      | None  |



# Programmed Request Summary

| <u>Mnemonic</u> | <u>Function</u>   | <u>Macro Call</u>              | <u>Error Codes<br/>(Byte 52=)</u>   |
|-----------------|---|--------------------------------|---|
| .PROTECT        | Used to obtain exclusive control of a vector in the range 0-476                                 | .PROTECT .area,.addr           | 0 - protect failure; locations already in use<br>1 - address > 476 or not a multiple of 4 |
| .PURGE          | Deactivates a channel without taking any other action   | .PURGE .chan                   | None  |
| .QSET           | Enlarges I/O queue for monitor  | .QSET .addr,.qleng             | None  |
| .RCTRLO         | Enables terminal printing   | .RCTRLO                        | None  |
| .RCVD           | Posts request to receive message and continues execution (F/B only)                             | .RCVD .area,.buff,.wcnt        | 0 - no other job exists in system   |
| .RCVDC          | Posts request to receive message and enters completion routine when message received (F/B only) | .RCVDC .area,.buff,.wcnt,.crtm | 0 - no other job exists in system   |
| .RCVDW          | Posts request to receive message and waits (F/B only) until received                            | .RCVDW .area,.buff,.addr,.wcnt | 0 - no other job exists in system   |



# Programmed Request Summary

| Mnemonic | Function   | Macro Call                                | Error Codes<br>(Byte 52=)   |
|----------|--|---|---|
| .READ    | Initiates transfer from specified channel to memory; returns to program immediately                      | .READ .area,.chan,.buff,.wcnt,.blk        | 0 - attempt to read past end-of-file<br>1 - hard error on channel<br>2 - channel not open |
| .READC   | Transfers words from specified channel to memory; returns control to specified routine when complete     | .READC .area,.chan,.buff,.wcnt,.crtn,.blk | 0 - attempt to read past end-of-file<br>1 - hard error on channel<br>2 - channel not open |
| .READW   | Transfers words from specified channel to memory; returns control to user program when transfer complete | .READW .area,.chan,.buff,.wcnt,.blk       | 0 - attempt to read past end-of-file<br>1 - hard error on channel<br>2 - channel not open |



# Programmed Request Summary

| <u>Mnemonic</u>    | <u>Function</u>  | <u>Macro Call</u>                   | <u>Error Codes<br/>(Byte 52=)</u>                                   |
|--------------------|--|-------------------------------------|---|
| <u>.REGDEF</u>     | Defines general registers R0-R5 SP,PC                                      | <u>.REGDEF</u>                      | None  |
| <u>.RELEAS</u>     | Removes device handler from memory   | <u>.RELEAS .devnam</u>              | 0 - handler name is illegal   |
| <u>.RENAME</u>     | Changes file name  | <u>.RENAME .area,.chan,.dbl</u>     | 0 - channel open  |
| <u>.REOPEN</u>     | Reassociates channel with file on which a SAVESTATUS was performed         | <u>.REOPEN .area,.chan,.cbl</u>     | 1 - file not found<br>0 - channel is in use                         |
| <u>.RSUM</u>       | Resumes job after it was suspended via .SPND                               | <u>.RSUM</u>                        | None  |
| <u>.SAVESTATUS</u> | Stores five words (containing data concerning file definition) into memory | <u>.SAVESTATUS .area,.chan,.cbl</u> | 0 - channel not associated with a file<br>1 - SAVESTATUS is illegal |
| <u>.SDAT</u>       | Initiates message transfer; returns control to user program immediately    | <u>.SDAT .area,.buff,.wcnt</u>      | 0 - no other job exists   |



# Programmed Request Summary

| <u>Mnemonic</u> | <u>Function</u>  | <u>Macro Call</u>              | <u>Error Codes<br/>(Byte 52=)</u>  |
|-----------------|--|--------------------------------|--|
| .SDATC          | Initiates message transfer; transfers control to specified routine when complete | .SDATC .area,.buff,.wcnt,.crtn | 0 - no other job exists  |
| .SDATW          | Initiates message transfer; returns control to user program when complete        | .SDATW .area,.buff,.wcnt       | 0 - no other job exists  |
| .SERR           | Inhibits fatal errors from aborting job  | .SERR                          | -1 - called USR from completion routine<br>-2 - no device handler<br>-3 - error doing directory I/O<br>-4 - FETCH error<br>-5 - error reading overlay<br>-6 - no more room for files in directory<br>-7 - illegal address<br>-10 - illegal channel number<br>-11 - illegal EMT |



# Programmed Request Summary

| <u>Mnemonic</u> | <u>Function</u>   | <u>Macro Call</u>                               | <u>Error Codes<br/>(Byte 52=)</u>    |
|-----------------|---|---|--------------------------------------|
| .SETTOP         | Requests additional memory for program  | .SETTOP .addr                                   | None                                 |
| .SFPA           | Sets user interrupt address for floating point processor exceptions   | .SFPA .area,.addr                               | None                                 |
| .SPFUN          | Provides special device functions to magtape and cassette   | .SPFUN .area,.chan,.code,.blff,.wcnt,.crtm,.blk | 0 - attempt to read past end-of-file |
|                 |   |   | 1 - hardware error on channel        |
|                 |   |   | 2 - channel not open                 |
| .SPND           | Suspends running job  | .SPND   | None                                 |
| .SRESET         | Resets certain areas of memory, dismisses device handlers brought in by FETCH, purges currently open files, resets to 16 I/O channels, queue to one element | .SRESET   | None                                 |



# Programmed Request Summary

| <u>Mnemonic</u> | <u>Function</u>  | <u>Macro Call</u>   | <u>Error Codes<br/>(Byte 52=)</u>   |
|-----------------|--|---------------------|---|
| .SYNCH          | Enables monitor programmed request from within an interrupt service routine        | SYNCH .area         | Monitor returns to location following .SYNCH if:<br><br>1. another .SYNCH specifying same 7-word block is pending<br>2. illegal job number was specified<br>3. job is not running |
| .TLOCK          | Attempts to gain ownership of USR; if unsuccessful, control returns with C bit set | .TLOCK              | 0 - USR is in use by another job  |
| .TRPSET         | Allows user job to intercept traps to 4 and 10                                     | .TRPSET .area,.addr | None  |
| .TTYIN          | Inputs character from terminal and waits until done                                | .TTYIN .char        | None  |
| .TTINR          | Inputs character from terminal   | .TTINR              | 0 - No characters available in ring buffer  |



# Programmed Request Summary

| <u>Mnemonic</u> | <u>Function</u>   | <u>Macro Call</u>                         | <u>Error Codes<br/>(Byte 52=)</u>                                     |
|-----------------|---|---|---|
| .TTOUTR         | Outputs character to terminal   | .TTOUTR                                   | 0 - Output ring buffer full   |
| .TTYOUT         | Outputs character to terminal and waits until done  | .TTYOUT .char                             | None  |
| .TWAIT          | Suspends the running job for a specified amount of time   | .TWAIT .area,.time                        | 0 - No queue element was available                                    |
| .UNLOCK         | Releases USR from memory  | .UNLOCK                                   | None  |
| ..V2..          | Enables macro expansions to occur in Version 2 format   | ..V2..                                    | None  |
| .WAIT           | Suspends program execution until all channel I/O is complete  | .WAIT .chan                               | 0 - channel not open<br>1 - hardware error                            |
| .WRITC          | Initiates transfer from memory to specified channel and returns to user program; when complete, passes control to specified routine | .WRITC .area,.chan,.buff,.wcnt,.crtm,.blk | 0 - end-of-file reached<br>1 - hardware error<br>2 - channel not open |



# Programmed Request Summary

## Programmed Request Summary

| <u>Mnemonic</u> | <u>Function</u>  | <u>Macro Call</u>                   | <u>Error Codes<br/>(Byte 52=)</u>                                     |
|-----------------|--|-------------------------------------|---|
| .WRITE          | Initiates transfer from memory to channel; returns control to user program immediately | .WRITE .area,.chan,.buff,.wcnt,.blk | 0 - end-of-file reached<br>1 - hardware error<br>2 - channel not open |
| .WRITW          | Transfers words from memory to channel; when complete, returns control to user program | .WRITW .area,.chan,.buff,.wcnt,.blk | 0 - end-of-file reached<br>1 - hardware error<br>2 - channel not open |



## APPENDIX F

### BASIC/RT-11 LANGUAGE SUMMARY

BASIC/RT-11 is a single-user conversational BASIC for use under the RT-11 system. While the BASIC language is one of the simplest computer languages to learn, it contains advanced techniques which the experienced programmer can use to perform more intricate manipulations or to express a problem more efficiently.

BASIC/RT-11 interfaces with the RT-11 monitor to provide powerful sequential and random-access file capabilities and allows the user to save and retrieve programs from peripheral devices. BASIC/RT-11 has provision for alphanumeric character string I/O and string variables (12K or larger systems) and allows user-defined functions and assembly language subroutine calls from user BASIC programs.

For details on using the BASIC language, and for instructions on running BASIC/RT-11, refer to the BASIC/RT-11 LANGUAGE REFERENCE MANUAL (DEC-11-LBACA-A-D). A summary of the BASIC/RT-11 commands and error messages is included here for quick reference.

#### F.1 BASIC/RT-11 STATEMENTS

The following summary of BASIC statements defines the general format for the statement and gives a brief explanation of its use.

CALL "function name" [(argument list)]  
Used to call assembly language user functions from a BASIC program.

CHAIN "file descriptor" [LINE number]  
Terminates execution of user program, loads and executes the specified program starting at the line number, if included.

CLOSE  $\left\{ \begin{array}{l} \text{VFn} \\ \text{\#n} \end{array} \right\}$   
Closes the logical file specified. If no file is specified, closes all files which are open.

DATA data list  
Used in conjunction with READ to input data into an executing program.



# BASIC/RT-11 Language Summary

|   |   |
|---|---|
| DEF FNletter (argument)=expression  | Defines a user function to be used in the program (letter is any alphabetic letter).  |
| DIM variable(n), variable(n,m),variable\$(n),variable\$(n,m)              | Reserves space for lists and tables according to subscripts specified after variable name.  |
| END   | Placed at the physical end of the program to terminate program execution.   |
| FOR variable = expression1 TO expression2 STEP expression3                | Sets up a loop to be executed the specified number of times.  |
| GOSUB line number   | Used to transfer control to a specified line of a subroutine.   |
| GO TO line number   | Used to unconditionally transfer control to other than the next sequential line in the program.   |
| IF expression rel.op. expression { THEN } line number                     | Used to conditionally transfer control to the specified line of the program.  |
| IF END #n { THEN } line number  | Used to test for end file on sequential input file #n.  |
| INPUT list  | Used to input data from the terminal keyboard or papertape reader.  |
| INPUT #expression: list   | Inputs from a sequential file or particular device.   |
| [LET] variable=expression   | Used to assign a value to the specified variable.   |
| [LET] VFn(i)=expression   | Used to set the value of a virtual memory file element.   |
| NEXT variable   | Placed at the end of a FOR loop to return control to the FOR statement.   |
| OPEN file [ FOR { INPUT } ] [(b)] AS FILE #n [DOUBLE BUF]                 | Opens a sequential file for input or output as specified. File may be of the form "dev:filnam,ext" or a scalar string variable. The number of blocks may be specified by b.                       |
| OPEN file [ FOR { INPUT } ] [(b)] AS FILE VFn x (dimension)=string length | Opens a virtual memory file for input or output. x represents the type of file: floating point (blank), integer (%), or character strings (\$). File may be of the form "dev:fil.ext" or a scalar |



## BASIC/RT-11 Language Summary

|                                    |   |
|------------------------------------|---|
|                                    | string variable. The number of blocks may be specified by b.  |
| OVERLAY "file descriptor"          | Used to overlay or merge the program currently in memory with a specified file, and continue execution. |
| PRINT list                         | Used to output data to the terminal. The list can contain expressions or text strings.                  |
| PRINT "text"                       | Used to print a message or a string of characters.  |
| PRINT #expression: expression list | Outputs to a particular output device, as specified in an OPEN statement.                               |
| PRINT TAB(x);                      | Used to space to the specified column.  |
| RANDOMIZE                          | Causes the random number generator to calculate different random numbers every time the program is run. |
| READ variable list                 | Used to assign the values listed in a DATA statement to the specified variables.                        |
| REM comment                        | Used to insert explanatory comments into a BASIC program.   |
| RESTORE                            | Used to reset data block pointer so the same data can be used again.                                    |
| RESTORE #n                         | Rewinds the input sequential file #n to the beginning.  |
| RETURN                             | Used to return program control to the statement following the last GOSUB statement.                     |
| STOP                               | Used at the logical end of the program to terminate execution.  |

### F.2 BASIC/RT-11 COMMANDS

The following key commands halt program execution, erase characters or delete lines.

| <u>Key</u> | <u>Explanation</u>  |
|------------|---|
| ALTMODE    | Deletes the entire current line. Echoes DELETED message (same as CTRL U). On some terminals the ESC key must be used.   |
| CTRL C     | Interrupts execution of a command or program and returns control to the RT-11 monitor. BASIC may be restarted without loss of the current program by using the monitor REENTER command. |



## BASIC/RT-11 Language Summary

|        |   |
|--------|---|
| CTRL O | Stops output to the terminal and returns BASIC to the READY message when program or command execution is completed.             |
| CTRL U | Deletes the entire current line. Echoes DELETED message (same as ALTMODE).  |
| ←      | (SHIFT O) Deletes the last character typed and echoes a backarrow (same as RUBOUT). On VT05 or LA30 use the underscore (—) key. |
| RUBOUT | Deletes the last character typed and echoes a backarrow (same as ←).  |

The following commands list, punch, erase, execute and save the program currently in memory.

| <u>Command</u>                 | <u>Explanation</u>  |
|--------------------------------|---|
| CLEAR                          | Sets the array and string buffers to nulls and zeroes.                                  |
| LIST                           | Prints the user program currently in memory on the terminal.                            |
| LIST line number               |   |
| LIST -line number              |   |
| LIST line number-[END]         |   |
| LIST line number-line number   | Types out the specified program line(s) on the terminal.                                |
| LISTNH line number             |   |
| LISTNH -line number            |   |
| LISTNH line number-[END]       |   |
| LISTNH line number-line number | Lists the lines associated with the specified numbers but does not print a header line. |
| NEW "filnam"                   | Does a SCRatch and sets the current program name to the one specified.                  |
| OLD "file"                     | Does a SCRatch and inputs the program from the specified file.                          |
| RENAME "filnam"                | Changes the current program name to the one specified.                                  |
| REPLACE "dev:filnam.ext"       | Replaces the specified file with the current program.                                   |
| RUN                            | Executes the program in memory.   |
| RUNNH                          | Executes the program in memory but does not print a header line.                        |
| SAVE "dev:filnam.ext"          | Outputs the program in memory as the specified file.                                    |



## BASIC/RT-11 Language Summary

SCRatch                      Erases the entire storage area.

### F.3 BASIC/RT-11 FUNCTIONS

The following functions perform standard mathematical operations in BASIC.

| <u>Name</u> | <u>Explanation</u>  |
|-------------|---|
| ABS(x)      | Returns the absolute value of x.  |
| ATN(x)      | Returns the arctangent of x as an angle in radians in the range + or - pi/2.        |
| BIN(x\$)    | Computes the integer value of a string of blanks (ignored), 1's and 0's.            |
| COS(x)      | Returns the cosine of x radians.  |
| EXP(x)      | Returns the value of e <sup>x</sup> where e=2.71828.                                |
| INT(x)      | Returns the greatest integer less than or equal to x.                               |
| LOG(x)      | Returns the natural logarithm of x.   |
| OCT(x\$)    | Computes an integer value from a string of blanks (ignored) and digits from 0 to 7. |
| RND(x)      | Returns a random number between 0 and 1.  |
| SGN(x)      | Returns a value indicating the sign of x.   |
| SIN(x)      | Returns the sine of x radians.  |
| SQR(x)      | Returns the square root of x.   |
| TAB(x)      | Causes the terminal type head to tab to column number x.                            |

The string functions are:

|                |   |
|----------------|---|
| ASC(x\$)       | Returns as a decimal number the seven-bit internal code for the one-character string (x\$).                 |
| CHR\$(x)       | Generates a one-character string having the ASCII value of x.   |
| DAT\$          | Returns the current date in the format 07-MAY-73.   |
| LEN(x\$)       | Returns the number of characters in the string (x\$).   |
| POS(x\$,y\$,z) | Searches for and returns the position of the first occurrence of y\$ in x\$ starting with the zth position. |



## BASIC/RT-11 Language Summary

|                |   |
|----------------|---|
| SEG\$(x\$,y,z) | Returns the string of characters in positions y through z in x\$. |
| STR\$(x)       | Returns the string which represents the numeric value of x.       |
| TRM\$(x\$)     | Returns x\$ without trailing blanks.                              |
| VAL(x\$)       | Returns the number represented by the string (x\$).               |

### F.4 BASIC/RT-11 ERROR MESSAGES

| <u>Abbrevia-<br/>tion</u> | <u>Message</u>                        | <u>Explanation</u>   |
|---------------------------|---------------------------------------|--|
| ?ARG                      | ARGUMENT ERROR AT LINE xxxxx          | Arguments in a function call do not match (in number or in type) the arguments defined for the function.   |
| ?ATL                      | ARRAYS TOO LARGE AT LINE xxxxx        | There is not enough room in the memory available for the arrays specified in the DIM statements.   |
| ?BDR                      | BAD DATA READ AT LINE xxxxx           | Item input from DATA statement list by READ statement is bad.  |
| ?BRT                      | BAD DATA-RETYPE FROM ERROR            | Item entered to input statement is bad.  |
| ?BSO                      | BUFFER STORAGE OVERFLOW at line xxxxx | Not enough room available in file buffers.   |
| ?DCE                      | DEVICE CHANNEL ERROR AT LINE xxxxx    | The device channel number specified for a sequential or virtual memory file is out of range (1-7), or an attempt was made to open a virtual memory file on a non-file structured device. |
| ?DNR                      | DEVICE NOT READY                      | An OLD command read a file which did not have any BASIC statements.  |
| ?DV0                      | DIVISION BY 0 AT LINE xxxxx           | Program attempted to divide some quantity by 0.  |



## BASIC/RT-11 Language Summary

|      |  |   |
|------|--|---|
| ?ETC | EXPRESSION TOO COMPLEX AT LINE xxxxx   | The expression being evaluated caused the stack to overflow (usually because the parentheses are nested too deeply). The degree of complexity that produces this error varies according to the amount of space available in the stack at the time. Breaking the statement into several simpler ones eliminates the error. |
| ?FDE | FILE DATA ERROR AT LINE xxxxx          | Tried to write an element on an integer virtual memory file outside the range (x)<32,768.   |
| ?FIO | FILE I/O ERROR AT LINE xxxxx           | An I/O error occurred. All files are automatically closed.  |
| ?FNF | FILE NOT FOUND AT LINE xxxxx           | The file requested was not found on the specified device.   |
| ?FNO | FILE NOT OPEN AT LINE xxxxx            | The sequential or virtual memory file referenced is not open.   |
| ?FTS | FILE TOO SHORT AT LINE xxxxx           | The sequential file space allocated to an output file is inadequate.  |
| ?FWN | FOR WITHOUT NEXT AT LINE xxxxx         | The program contains a FOR statement without a corresponding NEXT statement to terminate the loop.  |
| ?GND | GOSUBS NESTED TOO DEEPLY AT LINE xxxxx | Program GOSUBS nested to more than 20 levels.   |
| ?IDF | ILLEGAL DEF AT LINE xxxxx              | The DEF statement contains an error.  |
| ?IDM | ILLEGAL DIM AT LINE xxxxx              | Syntax error in a dimension statement.  |
| ?ILN | ILLEGAL NOW                            | An attempt was made to execute an INPUT statement in immediate mode.  |
| ?ILR | ILLEGAL READ AT LINE xxxxx             | Tried to open a write-only device for input or tried to read on a sequential file open for output.  |
| ?LTL | LINE TOO LONG                          | The line being typed is longer than 120 characters; the line buffer overflows.  |



?NBF NEXT BEFORE FOR AT LINE xxxxx

?NER NOT ENOUGH ROOM AT LINE xxxxx

?NPR NO PROGRAM

?NSM NUMBERS AND STRINGS MIXED AT LINE xxxxx

GOOD OUT OF DATA AT LINE xxxxx

70VF      OVERFLOW AT LINE xxxxx

?PTB      PROGRAM TOO BIG

?PWF POWER FAIL AT LINE xxxxx

```
?RBG      RETURN BEFORE GOSUB AT LINE xxxxx
```

?SOB SUBSCRIPT OUT OF BOUNDS AT LINE xxxxx

7SS0 STRING STORAGE OVERFLOW AT LINE xxxxx

?STL STRING TOO LONG AT LINE xxxxx

?SYN SYNTAX ERROR AT LINE xxxxx

?TLT      LINE TOO LONG TO TRANSLATE

F-8



## BASIC/RT-11 Language Summary

line just entered exceeds the area available for translation.

?UFN     UNDEFINED FUNCTION AT LINE xxxxx  
The function called was not defined by the program or was not loaded with BASIC.

?ULN     UNDEFINED LINE NUMBER AT LINE xxxxx  
The line number specified in an IF, GO TO or GOSUB statement does not exist anywhere in the program.

?WLO     WRITE LOCKOUT AT LINE xxxx  
Tried to open a read-only device for output, or tried to write on a sequential or virtual file opened for input only.

?↑ER     ↑ ERROR AT LINE xxxxx  
The program tried to compute the value A↑B, where A is less than 0 and B is not an integer. This produces a complex number which is not represented in BASIC.

### Function Errors

The following errors can occur when a function is called improperly.

?ARG     The argument used is the wrong type (for example, the argument was numeric and the function expected a string expression).

?SYN     The wrong number of arguments was used in a function, or the wrong character was used to separate them (for example, PRINT SIN(X,Y) produces a syntax error).

In addition, the functions give the errors listed below.

|               |      |  |
|---------------|------|--|
| FNa(...)      | ?UFN | The function a has not been defined (function cannot be defined by an immediate mode statement). |
| RND or RND(x) |      | No errors.   |
| SIN(x)        |      | No errors.   |
| COS(x)        |      | No errors.   |
| SQR(x)        | ?ARG | x is negative.   |
| ATN(x)        |      | No errors.   |
| EXP(x)        | ?↑ER | x is greater than 87.  |
| LOG(x)        | ?ARG | x is negative or 0.  |



## BASIC/RT-11 Language Summary

|                  |      |  |
|------------------|------|--|
| ABS(x)           |      | No errors.   |
| INT(x)           |      | No errors.   |
| SGN(x)           |      | No errors.   |
| TAB(x)           | ?ARG | x is not in the range $0 \leq x < 256$ .             |
| LEN(a\$)         |      | No errors.   |
| ASC(A\$)         | ?ARG | A\$ is not a string of length 1.                     |
| CHR\$(x)         | ?ARG | x is not in the range $0 \leq x < 256$ .             |
| DAT\$            |      | No errors.   |
| POS(a\$,b\$,n)   |      | No errors.   |
| SEG\$(a\$,n1,n2) |      | No errors.   |
| TRM\$(a\$)       |      | No errors.   |
| VAL(a\$)         | ?ARG | A\$ is not a valid numeric expression.               |
| STR\$(x)         |      | No errors.   |
| BIN(x\$)         | ?ARG | Character other than blank, 0 or 1 in string.        |
| OCT(x\$)         | ?ARG | Character other than blank or 0 through 7 in string. |



## APPENDIX G

### FORTRAN LANGUAGE SUMMARY

FORTRAN IV is a problem-solving language designed to allow scientists and engineers to express mathematical operations in a familiar format.

A FORTRAN source program is composed of a series of statements. Statements are usually made up of one- or two-word commands, which, when used in conjunction with data variables and constants, can be used to control program execution, assign values to variables and read and write data. The FORTRAN library contains routines to simplify mathematical functions such as computing sines, cosines, square roots, etc. The source program is translated by the FORTRAN compiler into a machine language program which, when linked with the FORTRAN object time routines, can be executed by the computer.

This appendix describes first how to run a FORTRAN program as a foreground job, and then summarizes briefly the RT-11 FORTRAN IV language, statements, and error messages. For details, refer to the PDP-11 FORTRAN LANGUAGE REFERENCE MANUAL (DEC-11-LFLRA-A-D) and THE RT-11 FORTRAN COMPILER AND OBJECT TIME SYSTEM USER'S MANUAL (DEC-11-LRFP-A-D).

#### G.1 RUNNING A FORTRAN PROGRAM IN THE FOREGROUND

Since the FORTRAN Object Time System needs work areas to perform some of its functions, the FRUN/N:x monitor command must be used to allocate the needed space when running a FORTRAN program as a foreground job. The following formula can be used to calculate the number (x) which must be designated to the /N option:

$$x = y + 21 \text{ octal}(N) + (R-210 \text{ octal}) + A*400 \text{ octal}$$

where:

Y = 264 octal.

N = the value of the /N FORTRAN compiler switch option (the compiler defaults this to 6 octal).

R = the value of the /R FORTRAN compiler switch option (the compiler defaults this to 210 octal, 136 decimal).



## FORTRAN Language Summary

A = the number of I/O buffers in simultaneous use during execution. (Console terminal I/O does not require any buffers. Each open logical unit typically requires one buffer.)

For example, to calculate x for a program (FILENA.REL) which uses only terminal I/O and allows the compiler to assign the standard defaults, set the formula as follows (all values are octal):

$$x = 264 + 21(6) + (210-210) + 0*400$$

432 octal words are required. The FORTRAN program is executed using the FRUN command as shown:

```
.FRUN FILENA.REL/N:432 <CR>
```

### NOTE

An exception can occur if the size of the linked FORTRAN program is less than the size of the USR (2K) and the USR is to be swapped. In this case, additional space must be allocated to allow the USR to successfully swap over the linked FORTRAN program. The additional space needed may be calculated by subtracting the size of the linked FORTRAN program from 10000 octal bytes (the size of the USR) and adding this result to the value of x previously calculated.

## G.2 FORTRAN CHARACTER SET

The characters that may appear in a FORTRAN statement are restricted to those listed below.

1. The letters A through Z
2. The numerals 0 through 9
3. The following "special" characters:

| <u>Character</u> | <u>Name</u>      |
|------------------|------------------|
|                  | Space or Blank   |
| =                | Equals           |
| +                | Plus             |
| -                | Minus            |
| *                | Asterisk         |
| /                | Slash            |
| (                | Left Parenthesis |



## FORTRAN Language Summary

|    |                   |
|----|-------------------|
| )  | Right Parenthesis |
| ,  | Comma             |
| .  | Decimal Point     |
| '  | Apostrophe        |
| "  | Double Quote      |
| \$ | Dollar Sign       |

Other printable characters may appear in a FORTRAN statement only as part of a Hollerith constant or alphanumeric literal. Any printable character may appear in a comment.

### G.3 EXPRESSION OPERATORS

Each group of operators is shown in order of descending precedence during execution.

| <u>Type</u>       | <u>Operator</u>  | <u>Operates Upon</u>   |
|-------------------|--|--|
| <u>Arithmetic</u> |  |  |
| **                | exponentiation   | numeric constants,<br>variables and<br>expressions   |
| *,/               | multiplication,<br>division,   |  |
| +, -              | addition, subtraction  |  |
| -                 | unary minus  |  |
| <u>Relational</u> |  |  |
| .GT.              | greater than   | variables,<br>constants,<br>and arithmetic<br>expressions (all<br>relational operators<br>have equal priority)       |
| .GE.              | greater than or<br>equal to  |  |
| .LT.              | less than  |  |
| .LE.              | less than or<br>equal to   |  |
| .EQ.              | equal to   |  |
| .NE.              | not equal to   |  |
| <u>Logical</u>    |  |  |
| .NOT.             | .NOT.A is true if and<br>only if A is false,<br>and is false if A is<br>true.                            | logical variables,<br>logical constants,<br>integer variables<br>and constants, logical<br>integers and expressions. |
| .AND.             | A.AND.B is true if<br>and only if A and B<br>are both true and is<br>false if either A or<br>B is false. |  |
| .OR.              | A.OR.B is true if and<br>only if either A or   |  |



## FORTRAN Language Summary

B or both A and B  
are true, and false  
if both A and B  
are false.

.EQV. A.EQV.B is true if and  
only if A and B  
are both true or if A  
and B are both false.

.XOR. A.XOR.B is true if and only  
if A is true and B is  
false or if B is true  
and A is false.

### G.4 SUMMARY OF FORTRAN STATEMENTS

The following is a summary of statements available in the PDP-11 FORTRAN IV language, including the general format for the statement and its effect.

| <u>Statement Formats</u>              | <u>Effect</u>  |
|---------------------------------------|--|
| <u>Arithmetic/Logical Assignment</u>  |  |
| variable=expression                   | The value of the arithmetic or logical expression is assigned to the variable.   |
| <u>Arithmetic Statement Functions</u> |  |
| name(var1,var2,...)=expression        | Creates a user-defined function having the variables (var) as dummy arguments. When referenced, the expression is evaluated using the actual arguments in the function call. |
| ACCEPT                                |  |
| ACCEPT f,list                         | Reads input from logical unit 5 (TT: is default); f is the format statement label and list is an optional data list.   |
| ASSIGN                                |  |
| ASSIGN n TO ivar                      | Assigns the statement number n to the integer variable ivar.   |



## FORTRAN Language Summary

### BACKSPACE

BACKSPACE u

The currently open file on logical unit number u is backspaced one record.

### BLOCK DATA

BLOCK DATA

Specifies the subprogram which follows as a BLOCK DATA subprogram.

### CALL

CALL name  
CALL name (arg1, arg2,...)

Calls the SUBROUTINE subprogram with the name specified, passing the actual arguments (arg) to replace the dummy arguments in the SUBROUTINE definition.

### COMMON

COMMON/namel/var1, var2,.../namel/var(j),var(k)

Reserves one or more blocks of storage space under the name specified to contain the variables (var) associated with that block name.

### CONTINUE

CONTINUE

Causes no processing.

### DATA

DATA var1, var2,.../val1, val2,.../

Causes elements in the list of values (val) to be initially stored in the corresponding elements of the list of variable names (var).

### DECODE

DECODE (c,f,v)list

Changes the elements in the list of variables from ASCII into the desired internal format; c is the number of characters, f is the format, and v is the name of an array.



## FORTRAN Language Summary

### DEFINE FILE

DEFINE FILE u(m,n,U,lvar),...

Defines the record structure of a disk or DECTape direct access file where u is the logical unit number, m is the number of fixed length records in the file, n is the length in words of a single record, U is a fixed argument, and lvar is the associated variable pointing to the next record.

### DIMENSION

DIMENSION s1,s2,...,sk

Reserves storage space for the specified array(s). Each s consists of an array variable name followed by up to seven integer constants, separated by commas and enclosed in parentheses.

### DO

DO n lvar = e1,e2,e3

1. Sets integer variable ivar = to expression (e1, e2, e3 must result in an integer quantity)
2. Executes statements through statement number n
3. Increments lvar = lvar+e3
4. If e3>0 and ivar <=e2, or e3<0 and ivar >=e2, goes back to 2 above

### END

END

Delimits a program unit.

### ENCODE

ENCODE (c,f,v)list

Changes the elements in the list of variables into ASCII format; c is the number of characters in the buffer, f is the format statement number, and v is the name of the array to be used as the resulting buffer.

### END FILE

END FILE u

The file currently open on logical unit number u is closed.



## FORTTRAN Language Summary

### EQUIVALENCE

EQUIVALENCE (var1,var2,...),(varj,vark,...)

Each of the variables within a set of parentheses is assigned the same storage location.

### EXTERNAL

EXTERNAL name1,name2,...

Informs the system that the names specified are those of FUNCTION or SUBROUTINE subprograms.

### FIND

FIND(u'r)

Positions the direct access file on logical unit number u to record r and sets the associated variable to record number r.

### FORMAT

n FORMAT(field specification,...)

Describes the format in which one or more records are to be transmitted; the statement number n must be specified (the field specifications may include the field descriptors, I, O, F, E, D, G, L, A, H, X, T, Q, and \$, delimited by field separators , and /).

### FUNCTION

FUNCTION name  
FUNCTION name (var1,var2,...)  
type FUNCTION name (var1,var2,...)

Begins a FUNCTION subprogram, indicating the program name and any dummy variable names (var). An optional type specification can be included.

### GOTO

#### Unconditional

GOTO n

Transfers control to statement number n.



## FORTRAN Language Summary

### Computed

GOTO (k1,k2,...,kn),e

Transfers control to the statement number  $k_i$  where  $i$  = value of expression  $e$ . If  $e < 1$  or  $e > n$  no transfer takes place.

### Assigned

GOTO ivar  
GOTO ivar, (k1,k2,...,kn)

Transfers control to the statement most recently associated with ivar by an ASSIGN statement.

### IF

#### Arithmetic

IF (expression) n1,n2,n3

Transfers control to statement number  $n$  depending upon the value of the expression. If the value of the expression is less than zero, transfers to  $n_1$ ; if the value of the expression is equal to zero, transfers to  $n_2$ , if the value of the expression is greater than zero, transfers to  $n_3$ .

#### Logical

IF (expression) statement

Executes the statement if the logical expression tests true.

### IMPLICIT

IMPLICIT type (a1,b1-b2,...),...

The elements  $a$  and  $b$  represent single (or a range of) letter(s) whose presence as the initial letter of a variable specifies the variable to be of that type, if that variable is not explicitly given a type.

### PAUSE

PAUSE  
PAUSE display

Suspends program execution and prints the octal constant, Hollerith constant, or alphanumeric literal display, if one is specified. Program execution is resumed by typing a carriage return.



## FORTRAN Language Summary

### PRINT

PRINT f,list

Writes output on logical unit 6 (LP: is default); f is the format statement label and list is an optional data list.

### READ

#### Formatted

READ(u,f)  
READ(u,f)list  
READ f,list

Reads at least one logical record from device u according to format specification f and assigns values to the variables in the list. Logical unit number 1 is used as default in the third form above.

#### Unformatted

READ(u)  
READ(u) list

Reads one logical record from device u, assigning values to the variables in the list.

#### Direct Access

READ(u'r)list

Reads from logical unit number u, record number r, and assigns values to the variables in the list.

#### Transfer of Control on Error

END=n  
ERR=m  
END=n,ERR=m

Optional elements in the READ statement list (e.g., READ(u,f, END=n, ERR=m) allowing control transfer on error conditions. If an end-of-file condition is detected and END=n is specified, execution continues at statement n. If a hardware I/O error occurs and ERR=m is specified, execution continues at statement m.



## FORTRAN Language Summary

### RETURN

RETURN

Returns control to the calling program from the current subprogram.

### REWIND

REWIND u

Logical unit number u is repositioned to the beginning of the currently opened file.

### STOP

STOP  
STOP display

Terminates program execution and prints the octal constant, Hollerith constant, or alphanumeric literal display, if one is specified.

### SUBROUTINE

SUBROUTINE name  
SUBROUTINE name (var1,var2,...)

Begins a SUBROUTINE subprogram, indicating the program name and any dummy variable names (var).

### TYPE

TYPE f,list

Writes output on logical unit 7 (TT: is default); f is the format statement label and list is an optional data list.

### Type Declarations

type var1,var2,...,vark

The variable names, (var), are assigned the specified data type in the program unit. type is one of:

INTEGER(\*2,\*4)  
REAL(\*4,\*8)  
DOUBLE PRECISION  
COMPLEX(\*8)  
LOGICAL(\*4,\*1)

The optional \* indicates that an explicit length is to be set for the variable; the number that follows is the length in bytes. The length must be one of the permitted values above.



## FORTRAN Language Summary

### WRITE

#### Formatted

WRITE (u,f)  
WRITE (u,f) list

Causes one or more logical records containing the values of the variables in the list to be written onto device u according to the format specification f.

#### Unformatted

WRITE (u)  
WRITE (u)list

Causes one logical record containing the values of the variables in the list to be written onto device u.

#### Direct Access

WRITE (u'r) list

Causes one logical record containing the values of the variables in the list to be written onto record r of the logical unit number u.

#### Transfer of Control on Error

END=n  
ERR=m  
END=n,ERR=m

Optional elements in the WRITE statement list (e.g., WRITE u,f, END=n, ERR=m) allowing control transfer on error conditions. If an end-of-file condition is detected and END=n is specified, execution continues at statement n. If a hardware I/O error occurs and ERR=m is specified, execution continues at statement m.

## G.5 COMPILER ERROR DIAGNOSTICS

The RT-11 FORTRAN Compiler, while reading and processing the FORTRAN source program, can detect syntax errors (or errors in general form) such as unmatched parentheses, illegal characters, unrecognizable key words, missing or illegal statement parameters. The number of errors detected during compilation is indicated on the console terminal by the message:

ERRORS DETECTED: n

The next three sections describe the initial phase and secondary phase error diagnostics and the fatal FORTRAN Compiler error diagnostics.



## FORTRAN Language Summary

### G.5.1 Errors Reported by the Initial Phase of the Compiler

These errors are reported in the source program listing; they are not reported if a source listing is not requested. Only errors which cause a statement to be aborted are tabulated in the ERRORS DETECTED count.

The error diagnostics are printed after the source statement to which they apply (the L error diagnostic is an exception). The general form of the diagnostic is as follows:

\*\*\*\*\* c

Where c is a code letter whose meaning is described below:

| <u>Code Letter</u> | <u>Description</u>   |
|--------------------|--|
| B                  | Columns 1-5 of a continuation line are not blank. (Columns 1-5 of a continuation line must be blank except for a possible 'D' in column 1); the columns are ignored and compilation continues.           |
| C                  | Illegal continuation. Comments cannot be continued and the first line of a program unit cannot be a continuation line. The line is ignored and compilation continues.                                    |
| E                  | Missing END statement. An END statement is supplied by the Compiler if end-of-file is encountered.   |
| H                  | Hollerith string or quoted literal string longer than 255 characters or longer than the remainder of the statement. The statement is aborted.  |
| I                  | Non-FORTRAN character used. The line contains a character that is not in the FORTRAN character set and is not in a Hollerith string or comment line. The character is ignored and compilation continues. |
| K                  | Illegal statement label definition. Illegal (non-numeric) character in statement label field. The illegal statement label is ignored and compilation continues.  |
| L                  | Line too long. There are more than 80 characters in a line; this diagnostic is issued before the line containing too many characters. The line is truncated to 80 characters and compilation continues.  |
| M                  | Multiply defined label. The label is ignored.  |
| P                  | Statement contains unbalanced parentheses. The statement is aborted.   |
| S                  | Syntax error (multiple equal signs, etc.). Statement not of the general FORTRAN statement form. The statement is aborted.  |



## FORTRAN Language Summary

U Statement could not be identified as any legal FORTRAN statement. The statement is aborted.

### G.5.2 Errors Reported by Secondary Phases of the Compiler

Those Compiler error diagnostics not reported by the initial phase of the Compiler will appear immediately after the source listing and before the storage map. The general form of the diagnostic is:

IN LINE nnnnn MSG#m text

Where nnnnn is the internal sequence number of the statement in question, m is an integer constant specifying the error number, and text is a short description of the error. The line at fault is replaced by a call to a routine which generates a run-time error (see Section G.5) if program control passes through the statement. All errors reported by the secondary phase of the compiler are tabulated in the ERRORS DETECTED count.

The notation \*\*\*\* signifies that a particular variable name or statement label will appear at that place in the text.

#### ADJUSTABLE DIMENSIONS ILLEGAL FOR ARRAY \*\*\*\*

All arrays must be dimensioned with integer constants except as specified in the RT-11 FORTRAN COMPILER AND OBJECT TIME SYSTEM USER'S MANUAL (DEC-11-LRFPA-A-D).

#### ARRAY \*\*\*\* HAS TOO MANY DIMENSIONS

An array can have up to seven dimensions.

#### ATTEMPT TO EXTEND COMMON BACKWARDS

While attempting to equivalence arrays in COMMON, an attempt was made to extend COMMON past the recognized beginning of COMMON storage.

#### COMMON BLOCK EXCEEDS MAXIMUM SIZE

An attempt was made to allocate more space to COMMON than is physically addressable (>32K words).

#### DANGLING OPERATOR

An operator (+, -, \*, /, etc.) is missing an operand.  
Example: (I=J+).

#### DEFECTIVE DOTTED KEYWORD

A dotted relational operator was not recognized. Also, possible misuse of decimal point.

#### DO TERMINATOR \*\*\*\* PRECEDES DO STATEMENT

The statement specified as the terminator of a DO loop must come after the DO statement.

#### EXPECTING LEFT PARENTHESIS AFTER \*\*\*\*

An array name or function name reference is not followed by a left parenthesis.

#### EXTRA CHARACTERS AT END OF STATEMENT

All the necessary information for a syntactically correct FORTRAN statement has been found on this line,



## FORTRAN Language Summary

but more information exists. Possibly due to inadvertent continuation signal on next line, or a missing comma.

### FLOATING CONSTANT TOO SMALL

A floating constant in an expression is too close to zero to be represented in the internal format. Use zero if possible.

### ILLEGAL ADJACENT OPERATOR

Two operators (\*,/, logical operators, etc.) are illegally placed next to each other. Example: I/\*J.

### ILLEGAL ELEMENT IN I/O LIST

An item, expression, or implied DO specifier in an I/O list is of illegal syntax.

### ILLEGAL DO STATEMENT TERMINATOR \*\*\*\*

A DO statement terminator must not be a GO TO, arithmetic IF, RETURN, or DO statement or logical IF containing one of these statements.

### ILLEGAL STATEMENT ON LOGICAL IF

The statement contained in a logical IF must not be another logical IF or DO statement.

### ILLEGAL TYPE FOR OPERATOR

An illegal variable type has been used with an exponentiation or logical operator.

### ILLEGAL USAGE OF OR MISSING LEFT PARENTHESIS

A left parenthesis was required but not found, or a variable reference or constant is illegally followed by a left parenthesis.

### INTEGER OVERFLOW

An integer constant or expression value must not fall outside the range -32767 to +32767.

### INVALID COMPLEX CONSTANT

A complex constant has been improperly formed.

### INVALID DIMENSIONS FOR ARRAY\*\*\*\*

An attempt was made while dimensioning an array to explicitly specify zero as one of the dimensions.

### INVALID DO TERMINATOR ORDERING AT LABEL \*\*\*\*

Do loops are incorrectly nested.

### INVALID EQUIVALENCE

Illegal equivalence, or equivalence that is contradictory to a previous equivalence.

### INVALID FORMAT SPECIFIER

A format specifier is not the label of a FORMAT statement or an array name.

### INVALID IMPLICIT RANGE SPECIFIER

Illegal implicit range specifier, i.e., non-alphabetic specifier, or specifier range is in reverse alphabetic order.



## FORTRAN Language Summary

### INVALID LOGICAL UNIT

A logical unit reference must be an integer variable or constant in the range 1 to 99.

### INVALID OCTAL CONSTANT

An octal constant is too large or contains a digit other than 0-7.

### INVALID OPTIONAL LENGTH SPECIFIER

A data type declaration optional length specifier is illegal. For example, REAL\*4 and REAL\*8 are legal, but REAL\*6 is not.

### INVALID RADIX50 CONSTANT

Illegal character detected in a RADIX50 constant.

### INVALID RECORD FORMAT

The third parenthetical argument in a DEFINE FILE statement must be the single character U.

### INVALID STATEMENT IN BLOCK DATA

It is illegal to have any executable or FORMAT statements in a BLOCK DATA Subprogram.

### INVALID STATEMENT LABEL REFERENCE

Reference has been made to a statement number that is of illegal construction. GO TO 999999 is illegal since the statement number is too long.

### INVALID SUBROUTINE OR FUNCTION NAME

A name used in a CALL statement or function reference is not valid. Example: use of an array name in a CALL statement subroutine name reference.

### INVALID TARGET FOR ASSIGNMENT

The left side of an arithmetic assignment statement is not a variable name or array element reference.

### INVALID TYPE SPECIFIER

An unrecognizable data type was used.

### INVALID USAGE OF FUNCTION OR SUBROUTINE NAME

A function name cannot appear in a DIMENSION, COMMON, DATA, EQUIVALENCE, or Data Type Declaration statement.

### INVALID VARIABLE NAME

A variable name is missing, contains an illegal character, or does not begin with an alphabetic character.

### LABEL ON DECLARATIVE STATEMENT

It is illegal to place a label on a declarative statement.

### MISSING ASSIGNMENT OPERATOR

The first operator seen in an arithmetic assignment statement was not an equal sign (=). Example: I+J=K.

### MISSING COMMA

The comma delimiter was expected but was not found. See the section of the FORTRAN Reference Manual that



## FORTRAN Language Summary

describes the general form of the statement in question.

### MISSING DELIMITER IN EXPRESSION

Two operands have been placed next to each other in an expression, with no operator between them.

### MISSING LABEL

A statement label was expected but not found. Example: ASSIGN J TO I. A valid statement label reference should precede 'TO' but does not.

### MISSING RIGHT PARENTHESIS

Expecting a right parenthesis but one was not found. Example: READ(5,100,). The first non-blank character after the format reference should be a right parenthesis but is not.

### MISSING QUOTATION MARK

In a FIND statement, the logical unit number and record number must be separated by a single quotation mark.

### MISSING VARIABLE

A variable was expected but not found. Example: ASSIGN 100 TO 1. A variable name should follow the 'TO' but does not.

### MISSING VARIABLE OR CONSTANT

Looking for an operand (variable or constant) but found a delimiter (comma, parenthesis, etc.). Example: WRITE(). A unit number should follow the open parenthesis, but a delimiter (close parenthesis) is encountered instead.

### MODES OF VARIABLE \*\*\*\* AND DATA ITEM DIFFER

The data type of each variable and its associated data list item must agree in a DATA Statement.

### MULTIPLE DECLARATION FOR VARIABLE \*\*\*\*

A variable cannot appear in more than one data type declaration statement or dimensioning statement.

### NUMBER IN FORMAT STATEMENT NOT IN RANGE

An integer constant in a FORMAT statement is greater than 255 or is zero.

### PARENTHESES NESTED TOO DEEPLY

Group repeats in a FORMAT Statement have been nested too deeply.

### P-SCALE FACTOR NOT IN RANGE -127 TO +127

P-scale factors must fall in the range -127 to +127.

### REFERENCE TO INCORRECT TYPE OF LABEL \*\*\*\*

A statement label reference that should be a label on a FORMAT statement is not such a label, or a statement label reference that should be a label on an executable statement is not such a label.

### REFERENCE TO UNDEFINED STATEMENT LABEL

A reference has been made to a statement number that has not been defined anywhere in the program unit.



## FORTRAN Language Summary

### STATEMENT MUST BE UNLABELED

A DATA, SUBROUTINE, FUNCTION, BLOCK DATA, arithmetic statement function definition, or declarative statement must not be labeled.

### STATEMENT TOO COMPLEX

An arithmetic statement function has more than 10 dummy arguments. Or the statement is too long to compile. Break it up into 2 or more smaller statements.

### SUBROUTINE OR FUNCTION STATEMENT MUST BE FIRST

A SUBROUTINE, FUNCTION or BLOCK DATA Statement, if present, must be the first statement in a program unit.

### SUBSCRIPT OF ARRAY \*\*\*\* NOT IN RANGE

Array subscripts that are constants or constant expressions are checked to see if they are within the bounds of the array's dimensions.

### SYNTAX ERROR

Check the general form of the statement with the general form outlined in the FORTRAN LANGUAGE REFERENCE MANUAL section that describes that type of statement.

### TARGET MUST BE ARRAY

The third argument in an ENCODE or DECODE statement must be an array name.

### SYNTAX ERROR IN INTEGER OR FLOATING CONSTANT

An integer or floating constant has been incorrectly formed. For example, 1.23.4 is an illegal floating constant because it contains two decimal points.

### UNLABELED FORMAT STATEMENT

All FORMAT Statements must be labeled.

### USAGE OF VARIABLE \*\*\*\* INVALID

An attempt was made to EXTERNAL a common variable, an array variable, or a dummy argument or an attempt was made to place in COMMON a dummy argument or external name.

### VARIABLE \*\*\*\* INVALID IN ADJUSTABLE DIMENSION

A variable used as an adjustable dimension must be an integer dummy argument in the subprogram unit.

### WRONG NUMBER OF SUBSCRIPTS FOR ARRAY \*\*\*\*

An array reference does not have the same number of subscripts as specified when the array was dimensioned.

## G.5.3 Warning Diagnostics

Warning diagnostics report conditions which are not true error conditions, but which may be potentially dangerous at execution time, or which may present compatibility problems with FORTRAN Compilers running on other PDP-11 Operating Systems. The warning diagnostics are normally suppressed, but may be enabled by use of the /W Compiler



## FORTRAN Language Summary

switch. The form and placement of the warning diagnostics are the same as those for the secondary phase error diagnostics except that the line number reference is replaced with [WARNING]. A listing of the warning diagnostics follows:

### ADJUSTABLE DIMENSIONS ILLEGAL FOR ARRAY \*\*\*\*

Adjustable arrays must be a dummy argument in a subprogram, and the adjustable dimensions must be integer dummy arguments in the subprogram. Any variation from this rule will cause a dimension of 1 to be used and this warning message to be issued.

### NON-STANDARD STATEMENT ORDERING

Although the RT-11 FORTRAN Compiler has less-restrictive statement-ordering requirements than those outlined in chapter 7 of the PDP-11 FORTRAN LANGUAGE REFERENCE MANUAL, non-adherence to the stricter requirements may cause error conditions on other FORTRAN Compilers.

### VARIABLE \*\*\*\* IS NOT WORD ALIGNED

Placing a non-LOGICAL\*1 variable or array after a LOGICAL\*1 variable or array in COMMON or equivalencing non-LOGICAL\*1 variables or arrays to logical\*1 variables or arrays may cause this condition. An attempt to reference the variable at runtime will cause an error condition.

### VARIABLE \*\*\*\* NAME EXCEEDS SIX CHARACTERS

A variable name of more than six characters was specified. The first six characters were used as the true variable name. Other FORTRAN Compilers may treat this as an error condition.

## G.5.4 Fatal Compiler Error Diagnostics

These diagnostics, which are sent directly to the terminal, report hardware error conditions and conditions which may require rewriting of the source program. The form of the diagnostic is:

FATAL ERROR n

where n is one of the following error codes:

| <u>Code</u> | <u>Meaning</u>  |
|-------------|---|
| C           | Constant subscript overflow. Too many constant subscripts have been employed in a statement.<br><br>SOLUTION - simplify the statement.  |
| O           | Unrecoverable error occurred while the Compiler was writing the object file (.OBJ). Possibly, insufficient output file space.<br><br>SOLUTION - rectify hardware problem, or allow more space for output. |



## FORTRAN Language Summary

- P Optimizer push down overflow - statement too complex, or too many common subexpressions occurred in one basic block of a program.

SOLUTION - simplify complex statements.

- R Unrecoverable hardware error occurred while the Compiler was reading source file.

SOLUTION - rectify hardware problem.

- S Subexpression stack overflow - statement too complex.

SOLUTION - simplify complex statements.

- T Memory Overflow.

SOLUTION - break up program into subprograms or compile on larger machine.

- W Unrecoverable error occurred while the Compiler was writing listing file. Possibly, listing file space is not large enough.

SOLUTION - rectify hardware problem, or allow more space for listing file.

- Y Code generation stack overflow - statement too complex.

SOLUTION - simplify complex statements.

- Z Compiler error.

SOLUTION - report this error to your local software support representative. Please include program listing.

### G.6 OBJECT TIME SYSTEM ERROR DIAGNOSTICS

The Object Time System detects certain I/O, arithmetic, and system failure error conditions and reports them on the user terminal. These error diagnostics are printed in either a long or a short form.

The short form of the message appears as:

?ERR nn

where nn is a decimal error-identification number. The long form of the message appears as:

?ERR nn text

where nn is a decimal error-identification number and text is a short error description.

The default message length is long. The short message error module may be included with the program by using the /I Linker switch (see Chapter 6). The module named \$SHORT should be included from the FORTRAN library.



## FORTRAN Language Summary

There are four classes of OTS error conditions. Each error condition is assigned to one of these classes. The error condition classification for the error codes 1-16 can be changed by using the System Subroutine SETERR. (Refer to DEC-11-LRFPA-A-D.) Error codes 0 and 20-66 should not be changed from their FATAL classification or undeterminable results will occur. The classifications are:

|         |  |
|---------|--|
| IGNORE  | The error is detected but no error message is sent to the terminal. Execution continues.   |
| WARNING | The error message is sent to the terminal and execution continues.   |
| FATAL   | The error message is sent to the terminal and execution is terminated.   |
| COUNT:n | The error message is sent to the terminal and execution continues until the nth occurrence of the error, at which time the error will be treated as FATAL. |

If a program is terminated by a fatal error condition, active files are closed.

The OTS error diagnostics are listed below along with the error type and a brief explanation where necessary.

| <u>Error number</u> | <u>Error type</u> | <u>Message</u>  |
|---------------------|-------------------|---|
| 0                   | FATAL             | NON-FORTRAN ERROR CALL<br>A TRAP has occurred with an unrecognizable error code.  |
| 1                   | FATAL             | INTEGER OVERFLOW<br>During an arithmetic operation, an integer's value has exceeded 32767 in magnitude.   |
| 2                   | FATAL             | INTEGER ZERO DIVIDE<br>During an integer mode arithmetic operation an attempt was made to divide by zero.   |
| 3                   | FATAL             | COMPILER GENERATED ERROR<br>An attempt is made to execute a FORTRAN statement in which the compiler has detected errors.  |
| 4                   | WARNING           | COMPUTED GO TO OUT OF RANGE<br>The integer variable or expression in a computed GO TO statement was less than 1 or greater than the number of statement label references in the list. Control is passed to the next executable statement. |
| 5                   | COUNT:3           | INPUT CONVERSION ERROR<br>During a formatted input operation an illegal character was detected in an input field. A value of 0 is returned.   |



# FORTRAN Language Summary

| <u>Error<br/>number</u> | <u>Error<br/>type</u> | <u>Message</u>  |
|-------------------------|-----------------------|---|
| 6                       | IGNORE                | OUTPUT CONVERSION ERROR<br>During a formatted output operation the value of a particular number could not be output in the specified field length without loss of significant digits. The field is filled with *'s. |
| 10                      | COUNT:3               | FLOATING OVERFLOW<br>During an arithmetic operation a real value has exceeded the largest representable real number. A value of 0 is returned.  |
| 11                      | IGNORE                | FLOATING UNDERFLOW<br>During an arithmetic operation a real value has become less than the smallest representable real number, and has been replaced with a value of zero.  |
| 12                      | FATAL                 | FLOATING ZERO DIVIDE<br>During a real mode arithmetic operation an attempt was made to divide by zero.  |
| 13                      | COUNT:3               | SQRT OF NEGATIVE NUMBER<br>An attempt was made to take the square root of a negative number. The result is replaced by zero.  |
| 14                      | FATAL                 | UNDEFINED EXPONENTIATION OPERATION<br>An attempt was made to perform an illegal exponentiation operation. For example -3.**.5 is illegal because the result would be an imaginary number.                           |
| 15                      | FATAL                 | LOG OF NEGATIVE NUMBER<br>An attempt was made to take the logarithm of a negative number.   |
| 16                      | FATAL                 | WRONG NUMBER OF ARGUMENTS<br>One of the FORTRAN Library functions, or System Subroutines which checks for such an occurrence, has been called with an improper number of arguments.                                 |

The following error diagnostics should not be changed from the FATAL classification by use of the System Subroutine SETERR:

|    |       |   |
|----|-------|---|
| 20 | FATAL | INVALID CHANNEL NUMBER<br>An illegal logical unit number has been specified in an I/O statement. A logical unit number must be an integer between 1 and 99.   |
| 21 | FATAL | NO AVAILABLE CHANNELS<br>An attempt was made to have too many devices simultaneously open for I/O. The maximum number of active channels is six by default, but this number may be changed at compile time. |



## FORTRAN Language Summary

- |    |       |   |
|----|-------|---|
| 22 | FATAL | INPUT RECORD TOO LONG<br>During an input operation a record was encountered that was longer than the maximum record length. The default maximum record length is 133 (decimal) bytes, but this maximum may be altered at compile time.                  |
| 23 | FATAL | HARDWARE I/O ERROR<br>A hardware error has been detected during an I/O operation.   |
| 24 | FATAL | ATTEMPT TO READ/WRITE PAST END OF FILE<br>If this occurs during a WRITE, more space is needed to contain the output file.   |
| 25 | FATAL | ATTEMPT TO READ AFTER WRITE<br>An attempt was made to read after writing on a file. A WRITE must be followed by a REWIND or BACKSPACE before a read operation can be performed.   |
| 26 | FATAL | RECURSIVE I/O NOT ALLOWED<br>An expression in the I/O list of a WRITE statement has caused initiation of another READ or WRITE operation. This can happen if a FUNCTION that performs I/O is referenced in an expression in a WRITE statement I/O list. |
| 27 | FATAL | ATTEMPT TO USE DEVICE NOT IN SYSTEM   |
| 28 | FATAL | OPEN FAILED FOR FILE<br>A file could not be found.  |
| 29 | FATAL | NO ROOM FOR DEVICE HANDLER<br>There is not enough free memory left to accommodate a specific device handler.  |
| 30 | FATAL | NO ROOM FOR BUFFERS<br>There is not enough free memory left to set up required I/O buffers.   |
| 31 | FATAL | NO AVAILABLE RT-11 CHANNEL<br>More than the maximum number of RT-11 channels, 15, were requested to be simultaneously opened for I/O.   |
| 32 | FATAL | FMTD-UNFMTD-RANDOM I/O TO SAME FILE<br>An attempt was made to perform any combination of formatted, unformatted, or random access I/O to the same file.   |
| 33 | FATAL | ATTEMPT TO READ PAST END OF RECORD<br>An attempt was made to read a larger record than actually exists in a file.   |



## FORTRAN Language Summary

|    |       |   |
|----|-------|---|
| 34 | FATAL | UNFMTD I/O TO TTY OR LPT<br>An attempt was made to perform an unformatted write operation on the terminal or line printer.  |
| 35 | FATAL | ATTEMPT TO OUTPUT TO READ ONLY FILE   |
| 36 | FATAL | BAD FILE SPECIFICATION STRING   |
| 37 | FATAL | RANDOM ACCESS READ/WRITE BEFORE DEFINE FILE<br>A random access READ or WRITE operation was attempted before a DEFINE FILE was performed.                                      |
| 38 | FATAL | RANDOM I/O NOT ALLOWED ON TTY OR LPT<br>Random access I/O was illegally attempted on the terminal or line printer.  |
| 39 | FATAL | RECORD LARGER THAN RECORD SIZE IN DEFINE FILE<br>A record was encountered that was larger than that specified in the DEFINE FILE statement for a random access file.          |
| 40 | FATAL | REQUEST FOR A BLOCK LARGER THAN 65535<br>An attempt was made to reference an absolute block address greater than 65535.   |
| 41 | FATAL | DEFINE FILE ATTEMPTED ON AN OPEN UNIT<br>An open file must be closed before another DEFINE FILE can be performed on that unit.  |
| 42 | FATAL | MEMORY OVERFLOW COMPILING OBJECT TIME FORMAT<br>The OTS has run out of free memory while scanning an array format that was generated at run time.                             |
| 43 | FATAL | SYNTAX ERROR IN OBJECT TIME FORMAT<br>A syntax error was encountered while the OTS was scanning an array format that was generated at run time.                               |
| 44 | FATAL | 2ND RECORD REQUEST IN ENCODE/DECODE<br>ENCODE and DECODE will operate only on a single record at a time.  |
| 45 | FATAL | INCOMPATIBLE VARIABLE AND FORMAT TYPES<br>An attempt was made to output a real variable with an integer field descriptor or an integer variable with a real field descriptor. |
| 46 | FATAL | INFINITE FORMAT LOOP<br>The format associated with an I/O statement that includes an I/O list has no field descriptors to use in transferring those variables.                |



## FORTRAN Language Summary

- 47            FATAL ATTEMPT TO STORE OUTSIDE PARTITION  
                 In an attempt to store data into a  
                 subscripted variable, the address  
                 calculated for the array element in  
                 question does not lie within the section  
                 of memory allocated to the job. The  
                 subscript in question is out-of-bounds.  
                 (Message is issued only when  
                 bounds-checking modules have been  
                 installed in FORLIB.)
- 60            FATAL        STACK OVERFLOWED  
                 The hardware stack has overflowed. Proper  
                 Traceback will be impaired.
- 61            FATAL        ILLEGAL MEMORY REFERENCE  
                 This may be any type of BUS error, most  
                 probably an illegal memory address  
                 reference.
- 62            FATAL        FORTRAN START FAIL  
                 The program has been loaded into memory  
                 but there was not enough free memory  
                 remaining for the OTS to initialize work  
                 space and buffers.
- 63            FATAL        ILLEGAL INSTRUCTION  
                 The program has attempted to execute an  
                 illegal instruction, e.g., floating point  
                 arithmetic instruction on a machine with  
                 no floating point hardware.



## APPENDIX H

### F/B PROGRAMMING AND DEVICE HANDLERS

#### H.1 F/B PROGRAMMING IN RT-11, VERSION 2

Certain programming conventions must be observed in RT-11, Version 2, which were not required in Version 1. These conventions are necessary to permit interrupt routines to function properly while running two jobs in the F/B environment. All Version 2 device handlers follow these conventions; the user is urged to consult the listings of the example handlers (Section H.3) as they illustrate some of the techniques discussed.

#### NOTE

Device handlers distributed with RT-11, Version 1, will not work properly with Version 2. Also, any user-written device handlers should be re-written to comply with the Version 2 conditions. Instructions for interfacing new handlers to RT-11 are provided in the RT-11 SOFTWARE SUPPORT MANUAL. (DEC-11-ORPGA-B-D). Section H.2 describes Version 2 device handlers.

The procedures described in this appendix are necessary and must be followed to prevent system failures when jobs are running under RT-11. If at any time a program which services its own interrupts (and does not follow the guidelines described here) is run with another job, the system may malfunction. Therefore, it is required that all programs follow the procedures that are indicated here.

##### H.1.1 Interrupt Priorities

The status word for each interrupt vector should be set such that when an interrupt occurs, the processor takes it at level 7. Thus, a device which has its vectors at 70 and 72 has location 70 set to its service routine; location 72 contains 340. The 340 causes the service routine to be entered with the processor set to inhibit any device interrupts.



## F/B Programming and Device Handlers

### H.1.2 Interrupt Service Routine

If the conventions outlined in Section H.1.1 are followed, when an interrupt occurs, the processor priority will be 7. The first task of the interrupt service routine is to lower the processor priority to the correct value. This can be done using the .INTEN macro call. The call is:

```
.INTEN .priority
or
.INTEN .priority,.pic
```

The .INTEN call is explained in Chapter 9, Programmed Requests. On return from the .INTEN call, the processor priority is set properly; registers 4 and 5 have been saved and can be used without the necessity of saving them again.

For example, a user device interrupts at processor priority 5:

```
DEVPRI=5
DEVINT: .INTEN DEVPRI      ;NOTE, NOT #DEVPRI
      .
      .
      .
      RTS PC
```

### H.1.3 Return from Interrupt Service

When an interrupt has been serviced, instead of issuing an RTI to return from the interrupt, the routine must exit with an RTS PC. This RTS PC returns control to the monitor, which then restores registers 4 and 5, and executes the RTI.

When a device handler has completed a transfer and is ready to return to the monitor via the internal monitor completion address. R4 must be pointing to the fifth word of the handler. It is no longer necessary to have R0 and R3 on the stack, as it was in Version 1. The user is urged to consult the example handlers at the end of this section, which explains the operation of the sample handlers.

### H.1.4 Issuing Programmed Requests at the Interrupt Level

Programmed requests from interrupt routines must be preceded by a .SYNCH call. This ensures that the proper job is running when the programmed request is issued. The .SYNCH call assumes that nothing has been pushed onto the stack by the user program between the .INTEN call and the .SYNCH call. On successful completion of a .SYNCH, R0 and R1 have been saved and are free to be used. R4 and R5 are no longer free, and should be saved and restored if they are to be used.



## F/B Programming and Device Handlers

### H.1.5 Setting Up Interrupt Vectors

Devices for which no RT-11 handler exists must be serviced by the user program. For example, no LPS device handler exists; to use an LPS, the user must write his own interrupt service routine. It is the responsibility of the program to set up the vector for devices such as this. The recommended procedure is not to simply move the service routine address and 340 into the desired vector; rather, this operation must be preceded by a .PROTECT macro call. The .PROTECT ensures that neither the other job nor the monitor already has control of that device. If the .PROTECT is successful, the vector can be initialized.

### H.1.6 Using .ASECT Directives

With some exceptions, user programs should not contain .ASECT directives. The Linker does not allow .ASECTs above 1000 when the /R switch is used; in general all .ASECTs below 1000 are ignored. The exceptions are:

|       |         |                         |
|-------|---------|-------------------------|
| 40    | 1 word  | program start address   |
| 42    | 1 word  | stack start address     |
| 44    | 1 word  | job status word         |
| 46    | 1 word  | USR swap address        |
| 34,36 | 2 words | TRAP instruction vector |

.ASECTs may be used to initialize any of the above locations. To initialize any other locations between 0-777, the user should first .PROTECT the appropriate words, and then move the correct values into place.

A job which contains overlays cannot have any .ASECTs at all, including those listed above. If overlays are to be used, the program start address should be set up as the argument for the .END statement; the other values can be initialized when the program is initiated.

Since .ASECTs into device vectors are not permitted, the user program must initialize vectors at runtime. To do this, execute a .PROTECT for the specified vector. If the .PROTECT is successful, it is safe to move values into the vectors. If the .PROTECT request fails, it is an indication that the vector is already in use by another job and that it is not safe to initialize the vector.

### H.1.7 Using .SETTOP

Proper use of .SETTOP is vital to preserve system integrity. Since RT-11 employs no hardware memory protection, the user must at all times observe the value returned in R0 from a .SETTOP request. It must never be assumed that the value requested by the program is the value returned by .SETTOP.



## F/B Programming and Device Handlers

### H.2 DEVICE HANDLERS

This section deals with the device handlers which are part of the RT-11 System, Version 2. Any device dependent information or general information required by the user is contained here. No mention of a handler implies that no special conditions must be met to use that device (all disks and DECTape are in this category, and therefore are not covered here).

#### Differences Between V1 and V2 Device Handlers

User-written device handlers must, in all cases, conform to the standard practices for Version 2. Since the last word of a device handler is used by the monitor, the user should be sure to include one extra word at the end of his program when indicating the size of the handler.

The differences between Version 1 and Version 2 handlers include the following:

#### A. Header Words

In Version 1 handlers, the third header word was taken to be the priority at which the device interrupt was taken. In Version 2, this word should be 340, indicating that the interrupt should be taken at priority level 7.

#### B. Entry Conditions

It is not necessary to save/restore registers when the handler is first entered, although to do so is not harmful. In Version 1, it was necessary to preserve R5 on first entering the handler.

#### C. Interrupt Handling

When an interrupt occurs, Version 2 handlers must execute an .INTEN request or its equivalent. (This was not necessary in Version 1.) On return from the .INTEN, R4 and R5 may be used as scratch registers. Device handlers may not do EMT requests without executing a .SYNCH request. (Refer to Chapter 9 for explanations of .INTEN and .SYNCH.)

The handler must return from an interrupt via an RTS PC rather than an RTI, as in Version 1. The .INTEN request essentially reenters the handler via a JSR PC, and thus the RTS PC returns control to the monitor, which executes the RTI when appropriate.

When the transfer is complete, the handler must exit to the monitor to terminate the transfer and/or enter a completion routine. When return is made to the monitor, R4 should point to the fifth word of the handler.

The user is urged to refer to the example handlers enclosed. The same general techniques can be used to interface user device handlers.



## F/B Programming and Device Handlers

### H.2.1 PR (High-Speed Paper Tape Reader)

Unlike the PR handler for Version 1, the Version 2 PR handler does not print a  $\uparrow$  on the terminal when it is entered for the first time. The tape must be in the reader when the command is issued, or an input error occurs. This prohibits any two-pass operations from being done using PR. For example, linking and assembling from PR will not work properly; an input error will occur when the second pass is initiated. The correct (and recommended) procedure is to use PIP to transfer the paper tape to disk or DECTape, and then perform the operation on the transferred file.

### H.2.2 TT (Handler for Console Terminal)

The console terminal may be used as a peripheral device by using the TT handler. Note that:

1. A  $\uparrow$  is typed when the handler is ready for input.
2. CTRL Z can be used to specify the end of input to TT. No carriage return is required after the CTRL Z. If CTRL Z is not typed, the TT handler accepts characters until the word count of the input request is satisfied.
3. CTRL O, struck while output is directed to TT, causes an entire output buffer to be ignored on output. This is somewhat different than the normal action of CTRL O while at the console.
4. A single CTRL C struck while typing input to TT causes a return to the monitor. If output is directed to TT, a double CTRL C is required to return to the monitor if F/B is running. If the S/J monitor is running, only a single CTRL C is required to terminate output.
5. The TT handler can be in use for only one job (foreground or background) at a time, and for only one function (input or output) at a time. The terminal communication for the job not using TT is not affected at all.
6. The user may type ahead to TT; the input ring buffer is emptied before the keyboard is referenced. The terminating CTRL Z may also be typed ahead.
7. If the main line code of a job is using TT for input, and a completion routine does a .TTYIN, typed characters will be passed unpredictably to the .TTYIN and TT. Therefore, this should not be done.
8. If a job sends a buffer to TT for output and then does a .TTYOUT or a .PRINT, the output from the latter is delayed until the handler completes its transfer. If a TT output operation is started when the monitor's terminal output ring buffer is not empty (i.e., before the print-ahead is complete), the handler supersedes the ring buffer until its transfer is complete.



## F/B Programming and Device Handlers

### H.2.3 CR (Card Readers)

The card reader handler can transfer data either as ASCII characters in DEC 026 or DEC 029 card codes (Section H.4), or as column images controlled by the SET command (see Chapter 2). In ASCII mode (SET CR NOIMAGE), invalid punch combinations are decoded as the error character 134(octal)--backslash. In IMAGE mode, no punch combination is invalid; each column is read as 12 bits of data right-justified in one word of the input buffer. The handler continues reading until the transfer word count is satisfied or until a standard end-of-file card is encountered (12-11-0-1-6-7-8-9 punch in column 1; the rest of the card is arbitrary). On end-of-file, the buffer is filled with zeroes and the request terminates successfully; the next input request from the card reader gives an end-of-file error. Note that if the transfer count is satisfied at a point which is not a card boundary, the next request continues from the middle of the card with no loss of information. If the input hopper is emptied before the transfer request is complete, the handler will loop until the hopper is reloaded and the "START" button on the reader is pressed again. The transfer will then continue until completion or until another hopper empty condition exists. End-of-file is not reported on the hopper empty condition. The handler will loop if the hopper empties during the transfer regardless of the status of the SET CR HANG/NOHANG option. No special action is required to use the card reader handler with the CM-11 mark sense card reader. The program must be aware of the fact that only 56 columns per card will be transferred if NOTRIM is in effect.

### H.2.4 MT/CT (Magtape (TU10/TM11) and Cassette (TA11))

These devices are new to RT-11 (Version 2) and have a file structure which differs from the common RT-11 structure. Each handler is capable of entirely supporting its own file structure and of allowing RT-11 users full access to the devices without being totally familiar with them.

H.2.4.1 General Characteristics - Both magtape and cassette can operate in two possible modes: "hardware" mode and "software" mode. These names refer to the type of operation which can be performed on the device at a given time. Software mode is the normal mode of operation and the mode used when accessing the device through any of the RT-11 system programs. In software mode, the handler automatically attends to file headers and uses a fixed record length to transfer data (256-word for magtape; 64-word for cassette).

Hardware mode allows the user to read or write any format desired, using any record size. In this mode, the word count is taken as the physical record size.

When the handlers are initially loaded by either the .FETCH programmed request or the .LOAD command, only software functions are permitted. To switch from software to hardware mode, either a rewind or a non-file structured lookup must be performed. (A non-file structured lookup is a lookup in which the first word of the filename is null.)



## F/B Programming and Device Handlers

In software mode, the following functions are permitted:

|            |   |  |
|------------|---|--|
| ENTER      | - | Open new file for output                           |
| LOOKUP     | - | Open existing file for input and/or output         |
| DELETE     | - | Delete an existing file on the device              |
| CLOSE      | - | Close a file which was opened with ENTER or LOOKUP |
| READ/WRITE | - | Perform data transfer requests                     |

In ENTER, LOOKUP, and DELETE, an optional file count parameter can be specified for use with magtape or cassette. Its meaning is as follows:

| <u>Count Argument</u> | <u>Meaning</u>   |
|-----------------------|--|
| =0                    | A rewind is done before the operation.   |
| >0                    | No rewind is done. The value of the count is taken as a limit of how many files to skip before performing the operation (e.g., a count of 2 will skip over, at most, one file. A count of 1 will not skip at all). |
| <0                    | A rewind is done. The complement of the switch value is then used as the limit.  |

If the file indicated in the request is located before the limit is exhausted, the search succeeds at that point.

As an example, consider:

```
.LOOKUP #AREA,#0,#PTR,#5
BCS     A1
.
.
AREA:   .BLKW   10.
PTR:    .RAD50  /MT0/
        .RAD50  /EXAMPLMAC/
```

In this case, the file count argument is +5, indicating that no rewind is to be done and that MT0 is to be searched for the indicated file (EXAMPL.MAC). If the file is not found after four files have been skipped, or if an end-of-tape occurs in that space, the search is stopped, and the tape is positioned either at the end-of-tape (EOT) or at the start of the fifth file. If the named file is found within the five files, the tape is positioned at its start. If the EOT is encountered first, an error is generated.

As another example:

```
.LOOKUP #AREA,#0,#PTR,#-5
```

This performs a rewind, and then uses a file count of five in the same way the above example does.



## F/B Programming and Device Handlers

### H.2.4.2 Handler Functions

#### 1. LOOKUP

If the filename (or the first word of the filename) is null, the operation is considered to be a non-file structured LOOKUP. This operation puts the handler into hardware mode. A rewind is automatically done in this case.

If the filename is not null, the handler tries to find the indicated file. LOOKUP uses the optional file count as illustrated above. Only software functions are allowed.

#### 2. DELETE

DELETE will delete a file of the indicated name from the device. DELETE also uses the file count argument, and can thus do a delete of a numbered file as well as a delete by name. When a file is deleted from MT or CT, an unused space is created there. However, it is not possible to reclaim that space, as it is when the device is random access. The unused spot will remain until the volume is re-initialized and rewritten.

#### 3. ENTER

ENTER creates a new file of the indicated name on the device. ENTER uses the optional file count, and can thus ENTER a file by name or by number. If ENTER by name is done, the handler deletes any files of the same name it finds in passing. If ENTER by number is done, the indicated number of files is skipped, and the tape is positioned at the start of the next file.

#### NOTE

Care must be used in performing numbered ENTERs, as it is possible to ENTER a file in the middle of existing files and thus destroy any files from the next file to the end of the tape.

It is also possible to create more than one file with the same name, since ENTER will only delete files of the same name it sees while passing down the tape. If an ENTER is done with a count greater than 0, no rewind is performed before the file is entered. If a file of the same name is present at an earlier spot on the tape, the handler cannot delete it. A non-file structured ENTER performs the same function as a non-file structured LOOKUP but does not rewind the tape. Since both functions allow



## F/B Programming and Device Handlers

writing to the tape without regard to the tape's file structure, they should be used with care on a file structured tape.

### 4. CLOSE

CLOSE terminates operations to a file on magtape or cassette and resets the handler to allow more LOOKUPS, ENTERS, or DELETES. If a CLOSE is not performed to an ENTERed file, the end-of-tape mark will be missing and no new files can be created on that volume. In this case, the last file on the tape must be rewritten and CLOSED to create a valid volume.

### 5. READ/WRITE

READ and WRITE are unique in that they can be done either in hardware or software mode. In software mode (file opened with LOOKUP or ENTER), records are written in a fixed size (256 words for magtape, 64 words for cassette). The word count specified in the operation is translated to the correct number of records. On a READ, the user buffer is filled with zeroes if the word count exceeds the amount of data available.

Following is a discussion of how the various parameters for READ/WRITE are used for magtape and cassette.

#### a. Block Number

For READ operations to magtape, the block number is used for random access (i.e., blocks need not be read sequentially from magtape). WRITE operations, however, disregard the block number and merely write the next sequential block. Block 0 on either READ or WRITE causes the device to rewind to the start of the file. A subsequent WRITE will destroy all previous output to that file.

For cassette, only sequential operations are performed. If the block number is 0, the cassette is rewound to the start of the file. Any other block number is disregarded.

#### b. Word Count

On a magtape READ, if the word count is 0, the block number is ignored and the next sequential record is read, no matter how big it is.

#### NOTE

Care must be taken when performing a magtape READ, because the READ will be done even if the record to be read is larger than the buffer allocated.



## F/B Programming and Device Handlers

On a magtape WRITE, if the word count is 0, one 256-word block is written.

If the word count for cassette is 0, the following conditions are possible:

If the block number is non-zero, the operation is actually a file-name seek. The block number is interpreted as the file count argument, as discussed in the above example of LOOKUP. The buffer address should point to the RAD50 of the device and file to be located. This feature essentially allows an asynchronous LOOKUP to be performed. The standard LOOKUP request does not return control to the user program if the tape has been positioned properly, whereas this asynchronous version will return control and interrupt when the file is positioned.

The user may then do a synchronous, positively numbered LOOKUP to the file just positioned, thus avoiding a long synchronous search of the tape.

If the block number is 0, a CRC (cyclical redundancy check) error occurs.

Following is a description of the allowed hardware mode functions for the handlers, as well as examples of how to call them. In general, special functions are called by using the .SPFUN request; examples of usage follow each function. The special functions require a channel number as an argument. The channel must initially be opened with a non-file structured LOOKUP which places the handler in hardware mode.

The general form of the .SPFUN request is:

.SPFUN .area,.chan,.code,.buff,.wcnt,.blk,.crtn

where:

.code is the function code to be performed.

The request format is:

R0 ⇒ .area:

|    |           |
|----|-----------|
| 32 | .chan     |
|    | .blk      |
|    | .buff     |
|    | .wcnt     |
|    | .code 377 |
|    | .crtn     |

### 1. Magtape Special Functions

- a. Rewind (Code = 373) - Rewinds the tape to the load point. This puts the unit into hardware mode (as does a non-file structured LOOKUP) where any of the other functions may be done.

Sample Macro Call:

.SPFUN #AREA,#0,#373



## F/B Programming and Device Handlers

The above performs a synchronous rewind on channel 0 (i.e., control will not return before the tape is positioned). An asynchronous rewind could be done with:

```
.SPFUN #AREA,#0,#373,,,,#CRTN ;REWIND MT, CHANNEL 0
```

where CRTN is a completion routine to be entered when the operation is finished. The other arguments are not required for this call.

- b. Write End-of-File (Code = 377) - This request writes an end-of-file mark, thus terminating a file.

Sample Macro Call:

```
.SPFUN #AREA,#0,#377 ;THIS IS THE SYNCHRONOUS FORM
```

The asynchronous form is:

```
.SPFUN #AREA,#0,#377,,,,#CRTN ;WRITE EOF ON  
;MT, CHANNEL 0
```

- c. Forward Space (Code = 376) - This function spaces forward the specified number of records and then stops. If the EOT or EOF is discovered before the count is exhausted, the tape stops there. Note that the number of records to space forward is contained in the word count argument, and the number passed should be the positive value of the desired number.

Sample Macro Call:

```
.SPFUN #AREA,#0,#376,,#2 ;SPACE FORWARD 2  
;RECORDS, CHANNEL 0
```

This spaces forward two records.

- d. Back Space (Code = 375) - This is similar to Forward Space except the tape is backed up by the indicated number of blocks. Again, the word count must be the positive value of the number to back up.

Sample Macro Call:

```
.SPFUN #AREA,#0,#375,,#2
```

This backs the tape up by two records.

- e. Write With Extended Gap (Code = 374) - This request is the same as any of the WRITE requests, except that a longer file gap is written between records. This can be used to get past bad spots on the tape.



## F/B Programming and Device Handlers

### Sample Macro Call:

```
.SPFUN #AREA,#0,#374,#BUFF,#100.,#0
```

This performs a synchronous write, while:

```
.SPFUN #AREA,#0,#374,#BUFF,#100.,#1,#CRTN
```

is asynchronous and goes to CRTN when the operation is complete.

- f. Offline (Code = 372) - This request sets the drive to an off-line state. The unit can only be set on-line by manual control.

### Sample Macro Call:

```
.SPFUN #AREA,#0,#372
```

Since this is an instantaneous function, no asynchronous forms are required.

## 2. Cassette Special Functions

With exceptions noted, these requests are identical to those involving magtape.

- a. Rewind (Code = 373) - This request is identical to the rewind request for magtape, except that unless a completion routine is specified, control does not return to the user until the rewind completes.
- b. Last File (Code = 377) - This request rewinds the cassette and positions it immediately before the sentinel file (logical end-of-tape). The macro call is the same as for magtape code 377.
- c. Last Block (Code = 376) - This request rewinds one record. See the magtape code 376 for a sample macro call.
- d. Next File (Code = 375) - This request spaces the cassette forward to the next file.
- e. Next Block (Code = 374) - This request spaces the cassette forward by one record.



## F/B Programming and Device Handlers

- f. Write File Gap (code = 372) - This request terminates a file written by the user program when in hardware mode.

Sample Macro Call:

```
.SPFUN #AREA,#0,#372
```

This writes a file gap synchronously, while:

```
.SPFUN #AREA,#0,#372,,,,#1
```

or

```
.SPFUN #AREA,#0,#372,,,#CRTN
```

performs asynchronous write file gap operations.



F/B Programming and Device Handlers  
H.3 EXAMPLE DEVICE HANDLERS

```

PP V02-01 5/1/74      RT-11 MACRO VM02-09 11-SEP-74 10:47:13 PAGE 1

1  .TITLE PP V02-01 5/1/74
2
3  ; RT-11 HIGH SPEED PAPER TAPE (PC11) PUNCH HANDLER
4  ;
5  ; DEC-11-ORTPA-C
6  ;
7  ; ABC/RGR
8  ;
9  ; MAY 1974
10 ;
11 ; COPYRIGHT (C) 1974
12 ;
13 ; DIGITAL EQUIPMENT CORPORATION
14 ; MAYNARD, MASSACHUSETTS 01754
15 ;
16 ; THE INFORMATION IN THIS DOCUMENT IS SUBJECT TO
17 ; CHANGE WITHOUT NOTICE AND SHOULD NOT BE CONSTRUED
18 ; AS A COMMITMENT BY DIGITAL EQUIPMENT CORPORATION.
19 ; DIGITAL ASSUMES NO RESPONSIBILITY FOR ANY ERRORS THAT
20 ; MAY APPEAR IN THIS DOCUMENT
21 ;
22 ; THIS SOFTWARE IS FURNISHED TO PURCHASER UNDER A
23 ; LICENSE FOR USE ON A SINGLE COMPUTER SYSTEM AND
24 ; CAN BE COPIED (WITH INCLUSION OF DIGITAL'S COPYRIGHT
25 ; NOTICE) ONLY FOR USE IN SUCH SYSTEM, EXCEPT AS MAY
26 ; OTHERWISE BE PROVIDED IN WRITING BY DIGITAL.
27 ;
28 ; DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE
29 ; OR RELIABILITY OF ITS SOFTWARE ON EQUIPMENT
30 ; WHICH IS NOT SUPPLIED BY DIGITAL.
31 ;

```



H-15



[illegible]



```

43 000140 010704
44 000142 062704
45 000146 013705
46 000152 000175
47 000156 000000
48 000156 000000
49
50 000160
51
52 000001

```

PC,R4  
#PPCQE=,R4  
#MONLOW,R5  
#OFFSET(R5)

IADDR OF NEXT Q ENTRY POINTER  
JUMP TO Q MANAGER  
POINTS TO COMMON ENTRY CODE

SINPTR: .WORD 0  
PPSIZE = .-LOADPT  
END

PP V02=01 5/1/74 RT-11 MACRO VM02=09 11-SEP-74 10:47:13 PAGE 3+

## SYMBOL TABLE

```

HDERR = 000001
PP 000012R 002
PPINT 000036R 002
PPVEC = 000074
R1 =X000001
SP =X000006
. ABS. 000000 000
PC11PP 000160 001
ERRORS DETECTED: 0
FREE CORE: 16009, WORDS

```

LOADPT 000000R 002 MONLOW= 000054  
PPB = 177556 PPCQE 000010R  
PPLQE 000006R 002 PPREAD 000102R  
PR4 = 000200 PR7 = 000340  
R2 =X000002 R3 =X000003  
SINPTR 000156R 002

OFFSET= 000270  
PPDONE 000134R 002  
PPS = 177554  
PS = 177776  
R4 =X000004

PC  
PPERR 000130R  
PPSIZE= 000160  
R0 =X000000  
R5 =X000005

BIN:PP,LST:PP=SRC:PP/C/N:TTM:END

PP V02=01 5/1/74 RT-11 MACRO VM02=09 11-SEP-74 10:47:13 PAGE S-1  
CROSS REFERENCE TABLE (CREF V01=02)

```

SINPTR 3=48#
HDERR 3=3 3=44
LOADPT 3=19# 3=40
MONLOW 3=2# 3=50
OFFSET 2=20# 3=20
PP 2=21# 3=46
PPB 3=10# 3=30#
PPCQE 2=15# 3=10
PPDONE 3=6# 3=28

```

3=50  
3=45  
3=23 3=44  
3=42#



PPERR 3-25 3-40#  
 PPINT 3-3 3-20#  
 PPLQE 3-5#  
 PPREAD 3-12 3-34#  
 PPS 2-14# 3-13#  
 PPSIZE 3-50#  
 PPVEC 2-16# 3-2  
 PR4 2-24# 3-4  
 PR7 2-23# 3-22  
 PS 2-22# 3-35

3-24 3-42\*  
 3-22 3-39  
 3-36 3-39  
 3-36\*

PR V02-02 JUNE 13, 1974 RT-11 MACRO VM02-09 11-SEP-74 10:47:06 PAGE 1

1  
 2  
 3  
 4  
 5  
 6  
 7  
 8  
 9  
 10  
 11  
 12  
 13  
 14  
 15  
 16  
 17  
 18  
 19  
 20  
 21  
 22  
 23  
 24  
 25  
 26  
 27  
 28  
 29  
 30  
 31

.TITLE PR V02-02 JUNE 13, 1974  
 ; RT-11 HIGH SPEED PAPER TAPE READER (PC11) HANDLER  
 ;  
 ; DEC-11-ORPHA-C  
 ;  
 ; ABC/ECB/RGB/EF  
 ;  
 ; MAY 1974  
 ;  
 ; COPYRIGHT (C) 1974  
 ;  
 ; DIGITAL EQUIPMENT CORPORATION  
 ; MAYNARD, MASSACHUSETTS 01754  
 ;  
 ; THE INFORMATION IN THIS DOCUMENT IS SUBJECT TO  
 ; CHANGE WITHOUT NOTICE AND SHOULD NOT BE CONSTRUED  
 ; AS A COMMITMENT BY DIGITAL EQUIPMENT CORPORATION.  
 ; DIGITAL ASSUMES NO RESPONSIBILITY FOR ANY ERRORS THAT  
 ; MAY APPEAR IN THIS DOCUMENT  
 ;  
 ; THIS SOFTWARE IS FURNISHED TO PURCHASER UNDER A  
 ; LICENSE FOR USE ON A SINGLE COMPUTER SYSTEM AND  
 ; CAN BE COPIED (WITH INCLUSION OF DIGITAL'S COPYRIGHT  
 ; NOTICE) ONLY FOR USE IN SUCH SYSTEM, EXCEPT AS MAY  
 ; OTHERWISE BE PROVIDED IN WRITING BY DIGITAL.  
 ;  
 ; DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE  
 ; OR RELIABILITY OF ITS SOFTWARE ON EQUIPMENT  
 ; WHICH IS NOT SUPPLIED BY DIGITAL.  
 ;



```

PR  V02-02 JUNE 13, 1974      RT-11 MACRO VM02-09  11-SEP-74 10:47:06 PAGE 2

      00000000
      00000000
      00000001
      00000002
      00000003
      00000004
      00000005
      00000006
      00000007

      .CSECT PC11PR
      R0=X0
      R1=X1
      R2=X2
      R3=X3
      R4=X4
      R5=X5
      SP=X6
      PC=X7

      J PAPER TAPE READER CONTROL REGISTERS
      PRS = 177550
      PRB = 177552
      PRVEC = 70

      PRG0 = 1
      PINT = 101
      J CONSTANTS FOR MONITOR COMMUNICATION
      HDERR = 1
      MONLOW = 54
      OFFSET = 270

      J CONTROL REGISTER
      J DATA REGISTER
      J READER VECTOR ADDR

      J READER ENABLE BIT
      J INTERRUPT ENABLE BIT AND GO BIT

      J HARD ERROR BIT
      J POINTER TO MONITOR BASE
      J POINTER TO Q MANAGER COMPLETION ENTRY
  
```



PR V02-02 JUNE 13, 1974

RT-11 MACRO VM02-09

11-SEP-74 10:47:06 PAGE 3

```

1
2
3 000000 000070
4 000002 000044
5 000004 000340
6 000006 000000
7 000010 000000
8
9
10
11 000012 016703 177772
12 000016 006363 000006
13 000022 103461
14 000024 001462
15 000026 005737 177550
16 000032 100455
17 000034 012737 000101 177550
18 000042 000207
19
20 000044 000436
21
22
23
24 000046 013746 000054
25 000052 004536
26 000054 000140
27 000056 016704 177726
28 000062 062704 000004
29 000066 005737 177550
30 000072 100413
31 000074 113774 177552 000000
32 000102 005224
33 000104 005314
34 000106 001415
35 000110 052737 000001 177550
36 000116 000207
37
38 000120 005214
39 000122 105074 000000
40 000126 005364 000002
41 000132 001372
42 000134 052774 020000 177772

/ LOAD POINT
/ ADDR OF INTERRUPT VECTOR
/ OFFSET TO INTERRUPT ENTRY
/ PRIORITY 7
/ POINTER TO LAST Q ENTRY
/ POINTER TO CURRENT Q ENTRY

PR: MOV
ASL
BCS
REQ
TST
RMI
MOV
RTS

PRCQE: PRCQE,R3
6(R3)
PRERR
SEEK
0#PRS
PRERR
#PINT,0#PRS
PC

/ ENTRY POINT
/ ABORT ENTRY FOR F/B

/ INTO SYSTEM STATE
/ RETURN AT LEVEL 4
/ R4 POINTS TO Q ENTRY
/ POINT R4 TO BUFFER ADDRESS
/ ANY ERRORS?
/ YES-TREAT AS EOF
/ PUT CHAR IN BUFFER
/ BUMP BUFFER POINTER
/ DECREASE BYTE COUNT
/ IF ZERO, WE ARE DONE
/ RE-ENABLE READER

/ CLEAR REMAINDER OF BUFFER
/ MORE
/ SET EOF BIT IN CHANNEL STATUS

```



```

43
44
45
46 000142
47 000142 042737 000101 177550
48
49 000150 010704
50 000152 062704 177636
51 000156 013705 000054
52 000162 000175 000270
53
54 000166 052753 000001
55 000172 010446
56 000174 010546
57 000176 004767 177740

/ OPERATION COMPLETE
PRABRT:
PRDNE: BIC #PINT, #PRS
MOV PC, R4
ADD #PRCQE, R4
MOV #MONLOW, R5
JMP #OFFSET(R5)
PRERR: RIS
SEEK: R4, -(SP)
MOV R5, -(SP)
JSR PC, PRDNE

```

```

/TURN OFF THE READER INTERRUPT
/IN CASE WE GET AN ERROR LATER
/GET ADDR OF CURRENT Q ENTRY
/TO MONITOR COMPLETION
/SET HARD ERROR BIT
/SAVE R4 AND R5
/SO WE CAN GO INTO COMPLT
/COMPLT WILL RTS PC BACK HERE

```

PR V02-02 JUNE 13, 1974 RT-11 MACRO VM02-09 11-SEP-74 10:47:06 PAGE 3+

```

58 000202 012605
59 000204 012604
60 000206 000207
61 000210 004767 000002
62 000214 000207
63 000216 010346
64 000220 000750
65
66 000001
.END

/FAKE AN INTERRUPT
/IO MGR EXPECTS R3 ON STACK
/AND COMPLETE OPERATION

```

PR V02-02 JUNE 13, 1974 RT-11 MACRO VM02-09 11-SEP-74 10:47:06 PAGE 3+

## SYMBOL TABLE

```

WDERR = 000001
PR 000012R 002
PREOF 000122R 002
PRLOE 000006R 002
R2 =X000002
SP =X000006
ABS. 000000 000
000000 001
PC11PR 000222 002
ERRORS DETECTED: 0
FREE CORE: 16616. WORDS
WDERR = 000101
PRDNE 000142R 002
PRINT 000046R 002
R1 =X000001
SEEK 000172R 002

```

BIN:PR, LST:PR=SRC:PR/C/N:TTM:END



# F/B Programming and Device Handlers

PR V02-02 JUNE 13, 1974 RT-11 MACRO VM02-09 11-SEP-74 10:47:06 PAGE 3-1  
CROSS REFERENCE TABLE (CREF V01-02 )

|        |       |       |       |       |
|--------|-------|-------|-------|-------|
| HDERR  | 3-4   | 3-50  |       |       |
| MONLOW | 2-21* | 3-54  |       |       |
| OFFSET | 2-22* | 3-24  | 3-51  |       |
| PINT   | 2-23* | 3-52  |       |       |
| PR     | 2-19* | 3-17  | 3-47  |       |
| PRABRT | 3-11* |       |       |       |
| PRB    | 3-20  | 3-46* |       |       |
| PRCOE  | 2-15* | 3-31  |       |       |
| PRDNE  | 3-7*  | 3-11  | 3-27  | 3-50  |
| PRE01  | 3-34  | 3-47* | 3-57  | 3-64  |
| PREOF  | 3-38* | 3-41  |       |       |
| PRERR  | 3-30  | 3-39* |       |       |
| PRGO   | 3-13  | 3-16  | 3-54* |       |
| PRINT  | 2-18* | 3-35  |       |       |
| PRLOE  | 3-4   | 3-24* |       |       |
| PRS    | 3-6*  |       |       |       |
| PRVEC  | 2-14* | 3-15  | 3-17* | 3-29  |
| SEEK   | 2-16* | 3-3   |       | 3-35* |
|        | 3-14  | 3-55* |       | 3-47* |



# F/B Programming and Device Handlers

## H.4 DEC 026/DEC 029 CARD CODE CONVERSION TABLE

Card codes in the following table are broken down by zone punch.  
Except where noted, all codes apply to both DEC 026 and DEC 029 punched cards.

Table H-1  
Card Code Conversions

| Zone                         | Digit     | Octal | Character | Name         |
|------------------------------|-----------|-------|-----------|--------------|
| none                         | none      | 040   |           | SPACE        |
|                              | 1         | 061   | 1         | digit 1      |
|                              | 2         | 062   | 2         | digit 2      |
|                              | 3         | 063   | 3         | digit 3      |
|                              | 4         | 064   | 4         | digit 4      |
|                              | 5         | 065   | 5         | digit 5      |
|                              | 6         | 066   | 6         | digit 6      |
|                              | 7         | 067   | 7         | digit 7      |
| 12<br>(DEC 029)<br>(DEC 026) | none      | 046   | &         | ampersand    |
|                              |           | 053   | +         | plus sign    |
|                              | 1         | 101   | A         | upper case A |
|                              | 2         | 102   | B         | upper case B |
|                              | 3         | 103   | C         | upper case C |
|                              | 4         | 104   | D         | upper case D |
|                              | 5         | 105   | E         | upper case E |
|                              | 6         | 106   | F         | upper case F |
| 11                           | 7         | 107   | G         | upper case G |
|                              | none      | 055   | -         | minus sign   |
|                              | 1         | 112   | J         | upper case J |
|                              | 2         | 113   | K         | upper case K |
|                              | 3         | 114   | L         | upper case L |
|                              | 4         | 115   | M         | upper case M |
|                              | 5         | 116   | N         | upper case N |
|                              | 6         | 117   | O         | upper case O |
| 0                            | 7         | 107   | P         | upper case P |
|                              | none      | 060   | 0         | digit 0      |
|                              | 1         | 057   | /         | slash        |
|                              | 2         | 123   | S         | upper case S |
|                              | 3         | 124   | T         | upper case T |
|                              | 4         | 125   | U         | upper case U |
|                              | 5         | 126   | V         | upper case V |
|                              | 6         | 127   | W         | upper case W |
| 8                            | 7         | 130   | X         | upper case X |
|                              | none      | 70    | 8         | digit 8      |
|                              | 1         | 140   | `         | accent grave |
|                              | 2         | 072   | :         | colon        |
|                              | (DEC 029) | 137   | +         | backarrow    |
|                              | (DEC 026) |       |           | (underscore) |
|                              | (DEC 029) | 043   | #         | number sign  |

(continued on next page)



# F/B Programming and Device Handlers

Table H-1 (Cont.)  
Card Code Conversions

| Zone      | Digit | Octal | Character | Name                 |
|-----------|-------|-------|-----------|----------------------|
| (DEC 026) | 4     | 075   | =         | equal sign           |
| (DEC 029) | 5     | 100   | @         | commercial "at"      |
| (DEC 026) |       | 047   | '         | single quote         |
|           |       | 136   | ↑         | uparrow              |
| (DEC 029) | 6     | 075   | =         | equal sign           |
| (DEC 026) |       | 047   | '         | single quote         |
| (DEC 029) | 7     | 042   | "         | double quote         |
| (DEC 026) |       | 134   | \         | backslash            |
| 9         | none  | 071   | 9         | digit 9              |
|           | 2     | 026   | ctrl - V  | SYN                  |
|           | 7     | 004   | ctrl - D  | EOT                  |
| 12-11     | none  | 174   |           | vertical bar         |
|           | 1     | 152   | j         | lower-case J         |
|           | 2     | 153   | k         | lower-case K         |
|           | 3     | 154   | l         | lower-case L         |
|           | 4     | 155   | m         | lower-case M         |
|           | 5     | 156   | n         | lower-case N         |
|           | 6     | 157   | o         | lower-case O         |
|           | 7     | 160   | p         | lower-case P         |
| 12-0      | none  | 173   | {         | open brace           |
|           | 1     | 141   | a         | lower-case A         |
|           | 2     | 142   | b         | lower-case B         |
|           | 3     | 143   | c         | lower-case C         |
|           | 4     | 144   | d         | lower-case D         |
|           | 5     | 145   | e         | lower-case E         |
|           | 6     | 146   | f         | lower-case F         |
|           | 7     | 147   | g         | lower-case G         |
| 12-8      | none  | 110   | H         | upper-case H         |
| (DEC 029) | 2     | 133   | [         | open square bracket  |
| (DEC 026) |       | 077   | ?         | question mark        |
|           | 3     | 056   | .         | period               |
| (DEC 029) | 4     | 074   | <         | open angle bracket   |
| (DEC 026) |       | 051   | )         | close parenthesis    |
| (DEC 029) | 5     | 050   | (         | open parenthesis     |
| (DEC 026) |       | 135   | ]         | close square bracket |
| (DEC 029) | 6     | 053   | +         | plus sign            |
| (DEC 026) |       | 074   | <         | open angle bracket   |
|           | 7     | 041   | !         | exclamation mark     |
| 12-9      | none  | 111   | I         | upper-case I         |
|           | 1     | 001   | ctrl - A  | SOH                  |
|           | 2     | 002   | ctrl - B  | STX                  |
|           | 3     | 003   | ctrl - C  | ETX                  |
|           | 5     | 011   | ctrl - I  | HT                   |
|           | 7     | 177   |           | DEL                  |

(continued on next page)



# F/B Programming and Device Handlers

Table H-1 (Cont.)  
Card Code Conversions

| Zone | Digit       | Octal | Character | Name                         |
|------|-------------|-------|-----------|------------------------------|
| 11-0 | none        | 175   | }         | close brace                  |
|      | 1           | 176   | ~         | tilde                        |
|      | 2           | 163   | s         | lower-case S                 |
|      | 3           | 164   | t         | lower-case T                 |
|      | 4           | 165   | u         | lower-case U                 |
|      | 5           | 166   | v         | lower-case V                 |
|      | 6           | 167   | w         | lower-case W                 |
|      | 7           | 170   | x         | lower-case X                 |
| 11-8 | none        | 121   | Q         | upper-case Q                 |
|      | (DEC 029) 2 | 135   | ]         | close square bracket         |
|      | (DEC 026)   | 072   | :         | colon                        |
|      | 3           | 044   | \$        | currency symbol              |
|      | 4           | 052   | *         | asterisk                     |
|      | (DEC 029) 5 | 051   | )         | close parenthesis            |
|      | (DEC 026)   | 133   | [         | open square bracket          |
|      | (DEC 029) 6 | 073   | ;         | semi-colon                   |
|      | (DEC 026)   | 076   | >         | close angle bracket          |
|      | (DEC 029) 7 | 136   | ↑         | uparrow                      |
| 11-9 | (DEC 026)   | 046   | &         | (circumflex)<br>ampersand    |
|      | none        | 122   | R         | upper-case R                 |
|      | 1           | 021   | ctrl - Q  | DC1                          |
|      | 2           | 022   | ctrl - R  | DC2                          |
|      | 3           | 023   | ctrl - S  | DC3                          |
|      | 6           | 010   | ctrl - H  | BS                           |
| 0-8  | none        | 131   | Y         | upper-case Y                 |
|      | (DEC 029) 2 | 134   | \         | backslash                    |
|      | (DEC 026)   | 073   | ;         | semi-colon                   |
|      | 3           | 054   | ,         | comma                        |
|      | (DEC 029) 4 | 045   | %         | percent sign                 |
|      | (DEC 026)   | 050   | (         | open parenthesis             |
|      | (DEC 029) 5 | 137   | ←         | backarrow                    |
|      | (DEC 026)   | 042   | "         | (underscore)<br>double quote |
|      | (DEC 029) 6 | 076   | >         | close angle bracket          |
|      | (DEC 026)   | 043   | #         | number sign                  |
| 0-9  | (DEC 029) 7 | 077   | ?         | question mark                |
|      | (DEC 026)   | 045   | %         | percent sign                 |
|      | none        | 132   | Z         | upper-case Z                 |
|      | 5           | 012   | ctrl - J  | LF                           |
|      | 6           | 027   | ctrl - W  | ETB                          |
|      | 7           | 033   |           | ESC                          |
| 9-8  | 4           | 024   | ctrl - T  | DC4                          |
|      | 5           | 025   | ctrl - U  | NAK                          |
|      | 7           | 032   | ctrl - Z  | SUB                          |

(continued on next page)



# F/B Programming and Device Handlers

Table H-1 (Cont.)  
Card Code Conversions

| Zone      | Digit | Octal | Character | Name         |
|-----------|-------|-------|-----------|--------------|
| 12-9-8    | 3     | 013   | ctrl - K  | VT           |
|           | 4     | 014   | ctrl - L  | FF           |
|           | 5     | 015   | ctrl - M  | CR           |
|           | 6     | 016   | ctrl - N  | SO           |
|           | 7     | 017   | ctrl - O  | SI           |
| 11-9-8    | none  | 030   | ctrl - X  | CAN          |
|           | 1     | 031   | ctrl - Y  | EM           |
|           | 4     | 034   | ctrl - \  | FS           |
|           | 5     | 035   | ctrl - ]  | GS           |
|           | 6     | 036   | ctrl - ↑  | RS           |
|           | 7     | 037   | ctrl - —  | US           |
|           |       |       |           |              |
| 0-9-8     | 5     | 005   | ctrl - E  | ENQ          |
|           | 6     | 006   | ctrl - F  | ACK          |
|           | 7     | 007   | ctrl - G  | BEL          |
| 12-0-8    | none  | 150   | h         | lower-case H |
| 12-0-9    | none  | 151   | i         | lower-case I |
| 12-11-8   | none  | 161   | q         | lower-case Q |
| 12-11-9   | none  | 162   | r         | lower-case R |
| 11-0-8    | none  | 171   | y         | lower-case Y |
| 11-0-9    | none  | 172   | z         | lower-case Z |
| 12-11-9-8 |       |       |           |              |
|           | 1     | 020   | ctrl - P  | DLE          |
| 12-0-9-8  |       |       |           |              |
|           | 1     | 000   |           | NUL          |



## APPENDIX I

### DUMP

RT-11 DUMP is a program which outputs to the console or lineprinter all or any part of a file in octal words, octal bytes, ASCII characters and/or RAD50 characters. DUMP is particularly useful for examining data such as directories or files.

#### I.1 CALLING AND USING DUMP

DUMP is called using the monitor command:

```
R DUMP
```

in response to the dot printed by the Keyboard Monitor. The name of the file which is to be output is entered as follows in response to the asterisk printed by the Command String Interpreter:

```
dev:output=dev:input/s
```

where:

|        |  |
|--------|--|
| dev:   | represents any valid device specification (line printer is default for output if no output file is designated).              |
| output | represents the filename and extension assigned to the output file. The default extension for file-structured output is .DMP. |
| input  | represents the input source filename and extension.  |
| /s     | represents one or more of the switches listed in Table I-1.  |

Type CTRL C to halt DUMP at any time and return control to the monitor. To restart DUMP type R DUMP or the REENTER command in response to the monitor's dot.



### I.1.1 DUMP Switches

Table I-1  
DUMP Switches

| Switch | Meaning   |
|--------|---|
| /B     | Output octal bytes                              |
| /E:n   | End output at block number n                    |
| /G     | Ignore input errors                             |
| /N     | Suppress ASCII output                           |
| /O:n   | Output only block number n (same as /E:n, /S:n) |
| /S:n   | Start output with block number n                |
| /W     | Output octal words                              |
| /X     | Output RAD50 characters                         |

If an input filename is given, block numbers are relative to the beginning of the file to which the block belongs. If not, block numbers are absolute block numbers on the device (i.e., the physical block numbers on the corresponding device).

The following are two examples of DUMP. /B is used in the first example to output octal bytes of the file SQRT.FTN into a file called DIF.DMP on device DT1.

If DIF.DMP is then listed on the line printer (using PIP), it appears as follows:

[illegible]



# DUMP

|      |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
|------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| 120/ | 016 | 000 | 004 | 000 | 007 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 346 | 001 | 000 | 016 |
| 140/ | 000 | 003 | 000 | 000 | 000 | 067 | 011 | 076 | 371 | 000 | 000 | 167 | 001 | 000 | 014 | 000 |
| 160/ | 004 | 000 | 004 | 006 | 054 | 253 | 100 | 070 | 226 | 000 | 000 | 000 | 000 | 000 | 000 | 000 |
| 200/ | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 |
| 220/ | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 |
| 240/ | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 |
| 260/ | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 |
| 300/ | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 |
| 320/ | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 |
| 340/ | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 |
| 360/ | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 |
| 400/ | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 |
| 420/ | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 |
| 440/ | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 |
| 460/ | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 |
| 500/ | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 |
| 520/ | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 |
| 540/ | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 |
| 560/ | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 |
| 600/ | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 |
| 620/ | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 |
| 640/ | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 |
| 660/ | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 |
| 700/ | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 |
| 720/ | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 |
| 740/ | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 |
| .    | .   | .   | .   | .   | .   | .   | .   | .   | .   | .   | .   | .   | .   | .   | .   | .   |

|      |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
|------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| 760/ | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 |
| .    | .   | .   | .   | .   | .   | .   | .   | .   | .   | .   | .   | .   | .   | .   | .   | .   |



# DUMP

The second example illustrates the use of /X to output RAD50 and octal values for locations in the file. The numbers in the left column represent the byte displacement. ASCII characters are printed in the far right hand column (dot represents a non-printing character):

R DUMP  
\*RK1:LINK0 FB/X

```
BLOCK NUMBER 0000
000/ 027011 044524 046124 004505 052122 044514 045516 051040 *.TITLE,RTLINK R*
    GNY K/L LHT ASM MSZ K/D LAB MEX
020/ 047517 020124 047503 042504 020040 054040 031460 030455 *OOT CODE X03-1*
    LSW EF6 LSK KCL EEX ND HGX G4/
040/ 006466 035412 051040 026524 030461 046040 047111 042513 *6.,; RT-11 LINKE*
    BOV IQ4 MEX GJD G43 LGH LUA KCS
060/ 006522 035412 042040 041505 030455 026461 051117 046114 *R.,; DEC-11-URLL*
    BEJ IQ4 J6 J0U G4/ GII MF1 LML
100/ 026501 026502 040514 005015 020073 040515 020131 032461 *A-B-LA.,; MAY 15*
    GIY GIZ J06 AXM EFK J07 EGA HTQ
120/ 020054 034461 032067 005015 020073 050105 020057 047105 *, 1974.,; EP/ EN*
    EE6 IFA HNG AXM EFK L3/ EE9 LT7
140/ 040510 041516 042105 041040 020131 043512 005015 005015 *HANCED BY JG....*
    J02 J00 J67 JWH EGA KPJ AXM AXM
160/ 006473 035412 041440 030117 051131 043511 052110 024040 */.,; COPYRIGHT (*
    B0S IQ4 J/X L39 MGA KPI MSP FP2
200/ 024503 024040 034461 032067 006451 035412 006440 035412 *C) (1974)..; ..)*
    FXC FP2 IFA HNG B0I IQ4 B0
220/ 042040 043511 052111 046101 042440 052521 050111 042515 * DIGITAL EQUIPME*
    J6 KPI MSQ LHA KBP MZA L33 KCU
240/ 052116 041440 051117 047520 040522 044524 047117 005015 *NT CORPORATION..*
    MSV J/X MF1 LSK JRB K/L LUG AXM
260/ 020073 040515 047131 051101 026104 046440 051501 040523 */ MAYNARD, MASSA*
    EFK J07 LUW MFG GCL LMX ML3 JRC
300/ 044103 051525 052105 051524 030040 033461 032065 005015 *CHUSETTS 01754..*
    KVS MMM MSM MML G. H3I HNE AXM
320/ 006473 035412 052040 042510 044440 043116 051117 040515 */.,; THE INFORMA*
    B0S IQ4 MRP KCP K. K18 MF1 J07
340/ 044524 047117 044440 020116 044124 051511 042040 041517 *TION IN THIS DOC*
    K/L LUG K. EF0 KV6 MMA J6 J01
360/ 046525 047105 020124 051511 051440 041125 042512 052103 *UMENT IS SUBJECT*
    LN7 LT7 EF6 MMA ML JXU KCR MSK
400/ 052040 006517 035412 041440 040510 043516 020105 044527 * TO.,; CHANGE WI*
    MRP BEG IQ4 J/X J02 KPN EFU K/O
420/ 044124 052517 020124 047516 044524 042503 040440 042116 *THOUT NOTICE AND*
    KV6 MY9 EF6 LSV K/L KCK JP2 J7F
440/ 051440 047510 046125 020104 047516 020124 042502 041440 * SHOULD NOT BE C*
    ML LSP LHU EFT LSV EF6 KCJ J/X
460/ 047117 052123 052522 042105 005015 020073 051501 040440 *UNSTRUED.,; AS A*
    LUG MSB MZB J67 AXM EFK ML3 JP2
500/ 041440 046517 044515 046524 047105 020124 054502 042040 * COMMITMENT BY D*
    J/X LN1 K/E LN6 LT7 EF6 NKJ J6
520/ 043511 040511 020124 050505 044525 046520 047105 020124 *IGIAT EQUIPMENT *
    KPI J03 EF6 M E K/M LN2 LT7 EF6
540/ 047503 050122 051117 052101 047511 027116 005015 020073 *CORPORATION.,; *
    LSK LAB MF1 MSI LSK GPN AXM EFK
560/ 042504 020103 051501 052523 042515 020123 047516 051040 *DEC ASSUMES NO R*
    KCL EF3 ML3 MZC KCU EF5 LSV MEX
600/ 051505 047520 051516 041111 046111 052111 020131 047506 *ESPONSIBILITY FO*
    ML7 LSK MMF JXI LMI MSQ EGA LSN
620/ 020122 047101 020131 051105 047522 051522 052040 040510 *R ANY ERRORS THA*
    EF4 LT3 EGA MFU LSK MMJ MRP J02
640/ 006524 035412 046440 054501 040440 050120 040505 020122 *T.,; MAY APPEAR *
    BEL IQ4 LMX NKI JP2 L4 JQ/ EF4
```



# DUMP

```

660/ 047111 052040 044510 020123 047504 052503 042515 052116 *IN THIS DOCUMENT*
    LUA   MRP   K/    EF5   LSL   MYS   KCU   MSV
700/ 006456 035412 005015 020073 044124 051511 051440 043117 *...).; THIS SOF*
    BDN   IQ4   AXM   EFK   KV6   MMA   ML    KI9
720/ 053524 051101 020105 051511 043040 051125 044516 044123 *TWARE IS FURNISH*
    M86   MFQ   EFU   MMA   KH2   MF7   K/F   KV5
740/ 042105 052040 020117 052520 041522 040510 042523 020122 *ED TO PURCHASER *
    J67   MRP   EF1   MZ    J04   JQ2   KCS   EF4

```

```

760/ 047125 042504 020122 006501 035412 046040 041511 047105 *UNDER A..; LICEN*
    LUM   KCL   EF4   B03   IQ4   LGH   J0Y   LT7

```

## I.2 DUMP ERROR MESSAGES

The following errors may occur when using DUMP:

| <u>Message</u> | <u>Meaning</u>   |
|----------------|--|
| ?IN ERR?       | A hardware error occurred while reading the input file and /G was not specified in the command line. |
| ?OUT ERR?      | A hardware error occurred while writing an output file, or the output device was full.               |
| ?LP NOT FND?   | A line printer handler is not available on the system.   |







## APPENDIX J

### FILEX

#### J.1 FILEX OVERVIEW

FILEX is a general file transfer program used to convert files among file-formatted devices for various operating systems. Transfers may be initiated between any file-structured RT-11 device and PDP-11 DOS/BATCH DECTape (as input or output), DOS/BATCH disk (as input only), RSTS-11 DECTape (as input or output), and DECsystem-10 DECTape (as input only). Files are transferred as 16-bit binary data. No processing is done on the data itself except that which is necessary to convert between various data representations.

"Wild-card" names (the \*.\* construction explained in Chapter 4) are permitted.

##### J.1.1 File Formats

FILEX can transfer files created by three different operating systems--RT-11, DECsystem-10, and DOS/BATCH (PDP-11 Disk Operating System/BATCH). Data formats that may be used in a transfer are ASCII, image, and packed image. Because the file structure for each system varies somewhat, switches are needed in the command line to indicate the file structures used and the data format conversion (if any) to be performed. These switches are defined in Section J.2.1. Devices are assumed to be in RT-11 file structure unless either a /S or /T switch (for DOS/BATCH and RSTS-11 or DECsystem-10 respectively) is indicated. If both input and output devices are RT-11 format (or are not file-structured), FILEX will operate like PIP.

#### J.2 CALLING AND USING FILEX

FILEX is called from the RT-11 system device by typing:

R FILEX

in response to the dot printed by the Keyboard Monitor. An asterisk is printed when FILEX is loaded and ready to accept command string input.



## FILEX

Type CTRL C to halt FILEX at any time and return control to the monitor. To restart FILEX, type R FILEX or the REENTER command in response to the monitor's dot.

### J.2.1 FILEX Switch Options

Table J-1 lists the switch options which are used for various FILEX operations. /S and /T must appear following the device and filename to which they apply; other switches may appear anywhere in the command line. These switches are explained in more detail following the table.

Table J-1  
FILEX Switch Options

| Switch | Meaning  |
|--------|--|
| /A     | Indicates a character-by-character ASCII transfer in which rubouts and nulls are deleted; when /T is also used, each PDP-10 word is assumed to contain five 7-bit ASCII bytes. (The transfer terminates upon reaching a ↑Z for compatibility with RSTS-11; the ↑Z is not transferred.)   |
| /D     | Deletes the named file from the device; valid only for DOS/BATCH and RSTS-11 DECTape.  |
| /F     | Causes a "fast" listing of the device directory by listing filenames only.   |
| /I     | Performs an image mode transfer; if the input is either DOS/BATCH, RSTS-11 or RT-11, this is a word-for-word transfer; if the input is from DECSYSTEM-10, /I indicates that the file resembles a file created on DECSYSTEM-10 by MACY11, MACX11, or LNKX11 with the /I switch: in this case each DECSYSTEM-10 36-bit word contains one PDP-11 8-bit byte in its low-order bits.  |
| /L     | Causes a complete listing of the device directory, including filenames, block lengths, and creation dates, to appear on the console terminal.  |
| /P     | Performs a packed image transfer; if the input is either DOS/BATCH, RSTS-11 or RT-11, this is a word-for-word transfer; if the input is from DECSYSTEM-10, /P indicates that the file resembles a file created on DECSYSTEM-10 by MACY11, MACX11, or LNKX11 with the /P switch, in which case each DECSYSTEM-10 36-bit word contains four PDP-11 8-bit bytes aligned on bits 0, 8, 18, and 26. This mode is assumed if no mode switch (/A, /I) is indicated in a command line. |
| /S     | Indicates the device is a DOS/BATCH (or RSTS-11) file-structured device; the switch must appear on the   |

(continued on next page)



FILEX

Table J-1 (Cont)  
FILEX Switch Options

| Switch | Meaning  |
|--------|--|
|        | same side of the command line as the device to which it applies.<br><br>NOTE<br><br>Neglecting to include the /S switch in a command line involving DOS/BATCH (or RSTS-11) causes unpredictable results. |
| /T     | Indicates the device is a DECsystem-10 file-structured device; the switch must appear on the same side of the command line as the device to which it applies.  |
| /V     | Types out version number of FILEX.   |
| /Z     | Zeroes the directory of the specified device in the proper format (valid only for DOS/BATCH and RSTS-11 DECTape).  |

J.2.2 Transferring Files Between RT-11 and DOS/BATCH (or RSTS-11)

File transfers may be initiated between file-structured devices used by RT-11 and the PDP-11 DOS/BATCH system. DECTape used under the RSTS-11 system is also legal as both input and output, since its format is identical to DOS/BATCH DECTape; see PDP-11 RESOURCE SHARING TIME-SHARING SYSTEM USER'S GUIDE, DEC-11-ORSUA-A-D. Input from DOS/BATCH may be either disk or DECTape; both linked and contiguous files are supported as input; the RT-11 device DK: is assumed if no device is indicated. If the input device is a DOS/BATCH disk, the user should specify a DOS/BATCH user identification code (called a UIC--refer to the PDP-11 DISK OPERATING SYSTEM MONITOR PROGRAMMER'S HANDBOOK, DEC-11-OMONA-A-D); this UIC then becomes default for all future transfers. If no UIC is specified, the current default UIC is used (see the description of UIC, following). Output to DOS/BATCH is limited to DECTape only. Any valid RT-11 file storage device may be used for either input or output in the transfer.

NOTE

An RT-11 DECTape can hold more information than a DOS/BATCH or RSTS-11 DECTape. Thus, caution should be observed in copying files from a full RT-11 tape to a DOS DECTape as some information may not transfer. In such a case an error message will be printed and the transfer will not be completed.

When a transfer from an RT-11 device to a DOS DECTape occurs, the block size of the file may increase. However, if the file is later transferred back to an RT-11 device, the block size does not decrease.

To transfer a file from a DOS/BATCH file-structured device (or RSTS-11 DECTape) to an RT-11 device, the command string format is:



## FILEX

\*dev:filnam.ext=dev:filnam.ext/S/s[UIC]

where:

- dev: = on the output side, any valid RT-11 device (if that device is not file-structured, filnam.ext may be omitted); on the input side, DOS/BATCH DECTape or disk, or RSTS-11 DECTape.
- filnam.ext = for output, any valid RT-11 filename and extension; for input, any valid DOS/BATCH (RSTS-11) filename and extension.
- /S = the switch from Table J-1 which designates a DOS/BATCH (RSTS-11) file-structured device. (This switch MUST be included in the command line.)
- /s = optionally any other switch (one only) from Table J-1 (in this case, /A is the only meaningful switch which could be chosen); /A indicates an ASCII transfer is to be performed in which nulls and rubouts are deleted; if /A is not used, the transfer will be performed word-for-word.
- [UIC] = the DOS/BATCH user identification code in the format [nnn,nnn] where nnn represents 1 to 3 octal digits less than or equal to 377 specifying first the user-group number, and then the individual user number; this code may be placed anywhere on the side of the command line containing the DOS/BATCH device; whenever a UIC is entered, it becomes the default for any further DOS/BATCH transfers; the initial default value is [1,1].

### NOTE

A UIC need not be indicated in any command line if accessing only DECTape since individual users do not "own" files on DECTape under DOS; no error will occur if a code is used, however.

To transfer files from an RT-11 storage device to a DOS/BATCH or RSTS-11 DECTape, the command line format is as follows:

\*DTn:filnam.ext/S/s=dev:filnam.ext

where:

- DTn: = a valid DOS/BATCH (RSTS-11) DECTape assignment (only DECTape is legal for output).
- filnam.ext = for output, any valid DOS/BATCH (RSTS-11) filename and extension; for input, any valid RT-11 filename and extension.
- /S = the switch from Table J-1 which designates a DOS/BATCH (RSTS-11) file-structured device. (This switch MUST be included in the command line.)



## FILEX

/s = optionally any other switch (one only) from Table J-1 (in this case, /A is the only meaningful switch which could be chosen); /A indicates an ASCII transfer is to be performed in which nulls and rubouts are deleted; if /A is not used, the transfer will be performed word-for-word.

dev: = any valid RT-11 device.

### Examples:

```
*DT2: SORT. ABC/S= SORT. ABC
```

This command instructs FILEX to transfer a file called SORT.ABC from the RT-11 system device to a DOS/BATCH (or RSTS-11) formatted DECTape on unit DT2.

```
*PP:=DT2:TYPE.FIL/S/A
```

This command allows a file to be transferred from DOS/BATCH (or RSTS-11) DECTape to the papertape punch under RT-11. The transfer is done in ASCII mode.

```
*DK:*. **RK1:MACR1.MAC/S[200,200]
```

This command causes the file MACR1.MAC from the DOS/BATCH disk on unit 1 which is stored under the UIC [200,200] to be transferred to the RT-11 device DK. [200,200] becomes the default UIC for any further DOS/BATCH operations.

### J.2.3 Transferring Files to RT-11 from DECsystem-10

Files may be transferred to RT-11 devices from a DECsystem-10 DECTape whenever a foreground job is not running (this restriction is due to the fact that when reading DECsystem-10 files, FILEX accesses the DECTape control registers directly rather than using the RT-11 DECTape control handler). Output may be to any valid RT-11 device; DECsystem-10 DECTape is the only valid input device. The format of the command line is:

```
*dev:filnam.ext=DTn:filnam.ext/T/s
```

#### where:

dev: = any valid RT-11 device; if that device is not file-structured, the filnam.ext may be omitted.

filnam.ext = for output, any valid RT-11 filename and extension; for input, any valid DECsystem-10 filename and extension (see NOTE below).

DTn: = a valid DECsystem-10 DECTape assignment (only DECTape is legal for input).

/T = the switch from Table J-1 which signifies a DECsystem-10 file-structured device. When using /T, especially with /A, the time of day clock will lose time. It should be examined with the monitor TIME command, and reset if necessary.



## FILEX

/s = any other switch from Table J-1 which specifies the mode of transfer. Only one additional switch may be indicated per transfer; /P is assumed if no other mode is specified.

### NOTE

RT-11 files may be indirectly converted to DECsystem-10 format by running RT-11 FILEX and converting the files to DOS formatted DECTape, and then running DECsystem-10 FILEX to read the DOS DECTape; however, it is currently not possible under RT-11 to convert files directly from RT-11 to DECsystem-10 format.

### Examples:

```
*DT2:STAND.LIS=DT1:STAND.LIS/T/A
```

The ASCII file STAND.LIS is converted from DECsystem-10 ASCII format to RT-11 ASCII format and stored under RT-11 on DECTape 2 as STAND.LIS.

### NOTE

Transfers from DECsystem-10 DECTape to RT-11 DECTape may cause an <UNUSED> block to appear after the file on the RT-11 device. This is a result of the method by which RT-11 handles the increased amount of information on a DECsystem-10 DECTape.

```
*SY:*.NEW=DT0:*.LIS/T
```

This command indicates that all files on DECsystem-10 DECTape 0 with the extension .LIS are to be transferred to the RT-11 system device (disk or DECTape) using the same filename and an extension of .NEW. The /P switch is the assumed transfer mode.

### J.2.4 Listing Directories

Device directories of any of the file-structured devices used in a FILEX transfer can be listed on the console terminal. The following FILEX command lines are used:

|           |                         |
|-----------|-------------------------|
| *dev:/L   | for RT-11               |
| *dev:/L/S | for DOS/BATCH (RSTS-11) |
| *DTn:/L/T | for DECsystem-10        |



## FILEX

where:

dev: = disk or DECTape (DK is assumed if no device is specified).

DTn: = DECsystem-10 DECTape.

/L = the switch from Table J-1 which indicates a listing is desired (/F may be substituted if a "fast listing" is preferred).

/S = the switch from Table J-1 which designates a DOS/BATCH or RSTS-11 file-structured device.

/T = the switch from Table J-1 which designates a DECsystem-10 file-structured device.

Examples:

```
*RK1:/L/S
BADB .SYS      1  22-JUL-74
MONLIB.CIL    175C 22-JUL-74
DU11 .PAL      45  24-JUL-74
VERIFY.LDA    67C 22-JUL-74
CILUS .LDA     39  22-JUL-74
```

This command lists the complete disk directory of the device RK1. The letter "C" following the file size on a DOS/BATCH or RSTS-11 directory listing indicates the file is a contiguous file.

```
*DT1:*.PAL/L/S
```

This command lists all files with the extension .PAL which are on drive 1.

```
*DT1:*.*/F/T
```

All files on DECsystem-10 formatted DECTape 1, regardless of filename or extension, are listed; a fast directory is requested (/F) in which only filenames are printed.

### J.2.5 Deleting Files from DOS/BATCH (RSTS-11) DECTapes

FILEX may be used to delete files from, and zero directories of, DOS/BATCH and RSTS-11 formatted DECTapes. The format of these command lines is:

\*dev:filnam.ext/S/D to delete a file

or

\*dev:/S/Z to zero a directory

dev:/Z ARE YOU SURE?

where:

dev: = DOS/BATCH or RSTS-11 DECTape.

filnam.ext = a valid DOS/BATCH (RSTS-11) filename and extension.



## FILEX

- /S = the switch from Table J-1 which designates that the device is DOS/BATCH (RSTS-11) file-structured.
- /D = the switch from Table J-1 which designates that a file is to be deleted.
- /Z = the switch from Table J-1 which designates that a directory is to be zeroed.

### Examples:

\*DT0:\* PAL/D/S

All files on DECTape 0 with the extension .PAL are deleted.

\*DT2:TABLE.OBJ/D/S

The file TABLE.OBJ is deleted from the DECTape on unit 2.

\*DT0:/Z/S  
DT0:/Z ARE YOU SURE ?Y

The DECTape on drive 0 is initialized in DOS/BATCH (RSTS-11) format so that it contains no files.

## J.3 ERROR MESSAGES

Following is a list of error messages that may occur under FILEX:

| <u>Message</u>               | <u>Meaning</u>  |
|------------------------------|---|
| ?FILNAM1.EXT ALREADY EXISTS? | An attempt was made to create the named file (filnam.ext) on a DOS DECTape when a file already existed under the name specified. Use /D to delete the file, and retry the transfer.   |
| ?BAD PPN?                    | The DOS/BATCH user identification code was not in the form [nnn,nnn], where each nnn is an octal number less than or equal to 377(octal).   |
| ?COR OVR?                    | There was insufficient main storage for buffers and input list expansion. Try copying files one at a time, without using the "wild-card" (*.*) construction on input.                 |
| ?DEV FULL?                   | There was no room in the directory for the filename or there was no room on the output device for the file (in which case the filename is not placed in the output device directory). |
| ?DIR ERR?                    | An error occurred while reading or looking up the directory of the input device, or the   |



## FILEX

|                   |  |
|-------------------|--|
|                   | input device does not have the proper file structure.  |
| ?FEATURE NOT IMP? | An operation was attempted which FILEX cannot perform (e.g., zeroing an RT-11 device).   |
| ?FG ACTIVE?       | An attempt was made to use /T when a foreground job was active. The transfer is not allowed until the foreground job is terminated and unloaded via UNLOAD FG.   |
| ?FIL NAM?         | The output filename was invalid or null.   |
| ?FIL NOT FND?     | The input file was not found, or the "wild-card" (*.*) construction matched none of the existing files.  |
| ?ILL CMD?         | The length of the command line exceeded 72 characters; the command line was not in proper CSI format; the UIC exceeded the allowed number of characters or [ was used without ]; a "wild-card" (*.*) construction was used on a non-file-structured device; no output or no input file was specified for a copy operation, or more than one filename construction (dev:filnam.ext) was specified on either side of the = (<) sign. |
| ?ILL DEV?         | The device handler was not found, an invalid device name was used, or one of the following was attempted:<br><br>RK or DT was not used for DOS/BATCH (RSTS-11) input in a copy operation;<br><br>DT was not used for DOS/BATCH (RSTS-11) output in a zero or delete operation;<br><br>DT was not used for DOS/BATCH (RSTS-11) output in a copy operation;<br><br>DT was not used for DECsystem-10 input in any operation.          |
| ?ILL SWT?         | An illegal switch was used in a command line (e.g., a switch not listed in Table J-1).   |
| ?IN ERR?          | A device error occurred on input.  |
| ?NO UFD?          | The specified UFD was not found on the DOS input disk.   |
| ?OUT ERR?         | A device error occurred on output.   |
| ?SWT ERR?         | An attempt was made to use more than one /S or /T switch in a command line (only one is allowed); an attempt was made to use more than one transfer mode switch (/I, /P, /A) or more than one operation switch (/D, /L, /F, /Z) in a command line (only one of each is allowed).   |







APPENDIX K  
SOURCE COMPARE (SRCCOM)

The RT-11 Source Compare program (SRCCOM) is used to compare two ASCII files and to output any differences to a specified output device. It is particularly useful when the two files are different versions of a single program, in which case SRCCOM prints all the editing changes which transpired between the two versions.

K.1 CALLING AND USING SRCCOM

To run SRCCOM type the command:

R SRCCOM

followed by a carriage return in response to the dot printed by the Keyboard Monitor; the CSI prints an asterisk. Then enter the names of the files which are to be compared using a command string in the following format:

dev:output=dev:input1,dev:input2/s

where:

dev:        is any valid device specification.

output     is the filename and extension assigned to the output file. If no output file is indicated, output is directed to the terminal.

input1... are the input source filenames and extensions to be compared.

/s         is one of the switches listed in Table K-1.

Source files are examined line by line for groups of lines which match. When a mismatch occurs, all differences are output until n successive lines in the first file are identical to n lines in the second file. The number (n) is a variable which the user can set with the /L switch.



## Source Compare

### K.1.1 Extensions

No default extension is assigned by SRCCOM to the output file. The default extension for an input file is .MAC, representing a source file in MACRO language.

### K.1.2 Switches

Command switches are generally placed at the end of the command string but may follow any filename in the string. The following switches can appear in the command string:

Table K-1  
SRCCOM Switches

| Switch | Meaning  |
|--------|--|
| /B     | Compare blank lines. Without this switch, blank lines are ignored.   |
| /C     | Ignore comments (all text on a line preceded by a semicolon) and spacing (spaces and tabs). This switch does not cause a line consisting entirely of a comment to become a blank line, and therefore ignored in the line count.  |
| /F     | Include form feeds in the output file. (Form feeds are still compared if /F is not used, but they are not included in the output of differences.)  |
| /H     | Type list of switches available (help text). No I/O device is necessary since /H always prints the help text on the terminal.  |
| /L:n   | Specify the number of lines that determines a match (n is an octal number <=310). All differences occurring before and after a match are output. In addition, the first line of the current match is output after the differences to aid in locating the place within each file at which the differences occurred. The default value for n is 3. |
| /S     | Ignore spaces and tabs.  |

### K.2 OUTPUT FORMAT

The first line of each file is always output as identification and is not compared. A blank line is then printed, followed by the differences between the files, in the following format:



# Source Compare

```

1)1      FILEA
1)      A
****
2)1      FILEB
2)      A

```

\*\*\*\*\*

% FILES ARE DIFFERENT

The different lines are listed followed by a reference line which is the same for both files. Note the example below.

The following example uses SRCCOM to compare an edited file and its backup version. The default value for a match is 3 lines. Blank lines are ignored but all other characters are compared.

Following the example is a coded explanation of the comparison.

```

R SRCCOM
*RK1:LINK0.FB, LINK0.BAK
A { 1)1      .TITLE  RTLINK ROOT CODE   X03-16
    2)1      .TITLE  RTLINK ROOT CODE   X03-15

B { 1)1      SEVENK= 31452                ; MINIMUM CORE TO START LINKER
    1)
    1)
C { 1)      .MCALL  .CSISPC, .CSIGEN, .SETTOP, .LOCK, .UNLOCK
    ****
    B { 2)1      SEVENK= 31500                ; JUST BELOW 8K RESIDENT
        2)
        2)
    C { 2)      .MCALL  .CSISPC, .CSIGEN, .SETTOP, .LOCK, .UNLOCK
        *****
    B { 1)2      .GLOBL  RSWIT, RELPTR, FBTXT, OVLNUM, RELOVL, RLSTRT
    C { 1)      .GLOBL  RELADR, PNRELO, RELID1, RSIZ1, OVSIZ1, OVLCD
        ****
    B { 2)2      .GLOBL  RSWIT, RELPTR, FBTXT, OVLNUM, RELOVL
    C { 2)      .GLOBL  RELADR, PNRELO, RELID1, RSIZ1, OVSIZ1, OVLCD
        *****
    B { 1)2      RLSTRT: .BLKW                ; CURRENT REL BLK OVERLAY NUM
    C { 1)      RELPTR: .BLKW                ; POINTER TO CURRENT REL BLK LOCATION
        ****
    C { 2)2      RELPTR: .BLKW                ; POINTER TO CURRENT REL BLK LOCATION
        *****
    B { 1)2      MTITLE: .ASCII /RT-11 LINK   X03-16/
    C { 1)      .ASCII  /   LOAD MAP  /
        ****
    B { 2)2      MTITLE: .ASCII /RT-11 LINK   X03-15/
    C { 2)      .ASCII  /   LOAD MAP  /
        *****
    B { 1)12     .IF DF FB
    B { 1)      MOV     OBLK, RLSTRT          ; IND START OF OVL FOR REL BLK
    B { 1)      .ENDC
    C { 1)      BR      1$
        ****
    C { 2)12     BR      1$
        *****

D { %FILES ARE DIFFERENT

```



## Source Compare

- A Headers, consisting of the first line of each file; for identification purposes.
- B n)m. A notation where n is the number of the input file, and m is the page number of the input file on which the text appears. The right column lists the lines in the files which are different.
- C Following a section of differences, a line identical to each file is output for reference purposes.
- D Indicates that the files are different (this is printed on the system console, not in the output file).

This example uses the /L:n switch and sets the number of lines that determines a match to 2 lines. The first two columns represent the input files:

```
TEST FILE 1
LINE C
LINE E
LINE C
LINE D
LINE F
LINE H
LINE I
LINE J
```

```
TEST FILE 2
LINE C
LINE D
LINE C
LINE E
LINE F
LINE G
LINE H
LINE I
LINE J
```

The files are compared and differences listed on the line printer.

```
*LP:=TEST1,TEST2/L:2
```

```
1)1    TEST FILE 1
2)1    TEST FILE 2
```



## Source Compare

```
1)1      LINE E
1)       LINE C
1)       LINE D
1)       LINE F
1)       LINE H
****
2)1      LINE D
2)       LINE C
2)       LINE E
2)       LINE F
2)       LINE G
2)       LINE H
*****
```

This message prints on the terminal indicating that the files are different.

%FILES ARE DIFFERENT

### K.3 ERROR MESSAGES

The following errors may be reported by SRCCOM:

| <u>Messages</u>       | <u>Meaning</u>   |
|-----------------------|--|
| ?COR OVR?             | Not enough memory to hold a particular difference section.               |
| ?IN ERR?              | A hardware error occurred in reading input.                              |
| ?OUT ERR?             | A hardware error occurred in writing output file, or output device full. |
| ?SWITCH ERROR?        | An invalid switch was found or a switch other than /L was given a value. |
| ?TOO MUCH DIFFERENCE? | More than 310 (octal) lines of difference between two files were found.  |







## APPENDIX L

### PATCH

The PATCH utility program is used to make code modifications to memory image (.SAV) files, including overlay-structured and monitor files. PATCH, like ODT, can be used to interrogate, and then to change, words or bytes in the file.

PATCH provides eight relocation registers. Before changing a program with PATCH, copy the old file to a backup file with PIP, as the old file is modified when PATCH is used.

#### L.1 CALLING AND USING PATCH

To run PATCH, type the command:

```
R PATCH
```

followed by the RETURN key in response to the dot printed by the monitor. PATCH prints a version number message:

```
PATCH V01-02
```

and then prints the message:

```
FILE NAME --  
*
```

In response to the asterisk, enter the name of the file to be modified, using the following format:

```
dev:filnam.ext/M/O
```

where:

|            |  |
|------------|--|
| dev:       | represents an optional device specification; if not specified, DK: is assumed.                             |
| filnam.ext | represents the name of the file which is to be patched, if an extension is not indicated, .SAV is assumed. |
| /M         | must be used if the file is an RT-11 monitor file.   |
| /O         | must be used if the file is an overlay-structured file.  |



## PATCH

After this information has been entered, the Command String Interpreter prints an asterisk indicating that it is ready to accept a command. Note that /O and /M, if used, must be specified when the file name is typed; they are not legal at any other time.

### L.2 PATCH COMMANDS

Table L-1 summarizes the PATCH commands.

Table L-1  
PATCH Commands

| Command  | Action   |
|----------|--|
| Vr;nR    | Set relocation register n to value Vr.                           |
| b:B      | Set bottom address of overlay file to b.                         |
| [s:]r,o/ | Open word location Vr + o in overlay segment s.                  |
| [s:]r,o\ | Open byte location Vr + o in overlay segment s.                  |
| <CR>     | Close currently open word/byte.                                  |
| <LF>     | Close currently open word/byte and open the next one.            |
| ↑ or ^   | Close currently open word/byte and open the previous one.        |
| @        | Close the currently open word and open the word addressed by it. |
| F        | Begin patching a new file.                                       |
| E        | Exit to RT-11 monitor.   |

Explanations of each command follow. An example of the use of the commands is provided in Section L.3.

#### L.2.1 Patch a New File

The F command causes PATCH to close the file being patched, and accept a new file name to be patched.



## PATCH

### L.2.2 Exit from PATCH

The E command causes PATCH to close the file being patched and return control to the RT-11 monitor.

### L.2.3 Examine, Change Locations in the File

For a non-overlay file, a word address may be opened, as with ODT, by typing:

```
[<relocation register>],offset/
```

At this point, PATCH will type out the contents of the location and wait for the user to type in either new location contents (in octal) or another command.

In an overlay file, the format is:

```
[<segment number>:][<relocation register>],offset/
```

Where <segment number> is the overlay segment number as it is printed on the link map for the file. If it is omitted, the root segment is assumed.

Similarly, to open a byte address in file, the format is:

```
[<relocation register>],offset\
```

for non overlay files, or

```
[<segment number>:][<relocation register>],offset\
```

for overlay files.

Once a location has been opened, the user may optionally type in the new contents in the format:

```
[<relocation register>],value
```

followed by one of these control characters:

|                   |   |
|-------------------|---|
| <carriage return> | Close the current location by changing its contents to the new contents (if specified), and await more control input. |
| <line feed>       | Close the current location, and open the next word/byte.  |
| ↑ or ^            | Close the current location, and open the previous word/byte.  |
| @                 | Close the current word location, and open the word addressed by it (in the same segment if an overlay file).          |



## PATCH

### L.2.4 Set Bottom Address

To patch an overlay file, PATCH must know the bottom address at which the program was linked if it is different from the initial stack pointer. This is the case if the program sets location 42 in an .ASECT. To set the bottom address, type:

<bottom address>;B

Note that the B command must be issued before any locations are opened for modification.

### L.2.5 Set Relocation Registers

The relocation registers 0-7 are set, as with ODT, by the R command. The R command has the format:

<relocation value>;<relocation register>R

Once one of the eight relocation registers has been set, the expression:

<relocation register>,<octal number>

typed as part of a command will have the value:

<relocation value> + <octal number>

## L.3 EXAMPLES USING PATCH

The following example shows how to patch a non-overlaid file. Assume the following program (EXAM):

```
.MAIN. RT=11 MACRO VM02=08 PAGE 1

1
2
3      000015      CR=    13
4      000012      LF=    12
5      000000'      .CSECT MAIN
6                      .MCALL .PRINT,.EXIT
7                      .NLIST BEX
8 000000      124 MSG1 .ASCII /THIS IS A SUCCESSFUL PATCH/<CR><LF>
9                      .LIST BEX
10 00034 000403 START: BR    EXIT
11 00036                      .PRINT MSG
12 00044      EXIT:   .EXIT
13      000034'      .END    START
```

This program has been assembled with MACRO and linked with LINK; execution causes no output of text:

R EXAM



## PATCH

To make a line of text print on the terminal, PATCH is used as follows:

```
.R PATCH
PATCH V01-02
FILE NAME--
*EXAM.SAV
*1000;0R
*0,34/ 403      240
*E
```

Now when the program is executed:

```
.R EXAM
THIS IS A SUCCESSFUL PATCH
```

The next example demonstrates a similar situation, only includes an overlay file. These programs have been assembled and linked; the output of both operations is included:

.MAIN. RT=11 MACRO VM02=08 3-SEP-74 PAGE 1

```
1
2
3      000015      CR#      15
4      000012      LFR#     12
5      000007      PC#      X7
6      000000'     .CSECT   MAIN
7                        .GLOBL ENTRY,MSG1
8                        .MCALL .PRINT,.EXIT
9                        .NLIST BEX
10 000000      124 MSG1 .ASCIZ /THIS IS A SUCCESSFUL PATCH/<CR><LF>
11 000035      124 MSG1 .ASCIZ /THIS IS AN OVERLAY PATCH/
12                        .LIST   BEX
13 000066 000403 START: BK      EXIT
14 000070                        .PRINT #MSG
15 000076 004767 EXIT: JSR      PC,ENTRY
                        000000G
16 00102                        .EXIT
17      000066'     .END      START
```



,MAIN. RT-11 MACRO VM02-08 3-SEP-74 PAGE 1

```

1
2
3      000015      CR=      15
4      000012      LF=      12
5      000007      PC=      X7
6      000020'      .CSECT  OVL
7                      .MCALL .PRINT
8                      .GLOBL MSG1
9                      .GLOBL ENTRY
10 00000 000403 ENTRY: BR      RETURN
11 00002                      .PRINT #MSG1
12 00010 000207 RETURN: RTS    PC
13      000001'      .END

```

RT-11 LINK X03-18 LOAD MAP  
PTCH .SAV 03-SEP-74

| SECTION        | ADDR   | SIZE   | ENTRY   | ADDR   | ENTRY | ADDR | ENTRY | ADDR |
|----------------|--------|--------|---------|--------|-------|------|-------|------|
| .ABS.          | 000000 | 001122 |         |        |       |      |       |      |
| MAIN           | 001122 | 000104 | MSG1    | 001157 |       |      |       |      |
| OVERLAY REGION | 000001 |        | SEGMENT | 000001 |       |      |       |      |
| OVL            | 001230 | 000012 | ENTRY   | 001230 |       |      |       |      |

TRANSFER ADDRESS = 001210  
HIGH LIMIT = 001242

Running the program (PTCH) produces no terminal output:

R PTCH

But by using PATCH to modify the file as follows:



## PATCH

. R PATCH

PATCH V01-02

FILE NAME--

\*PTCH. SAV/O

\*1230;OR

\*1:0,0/ 403 240

\*E

the following line results:

R PTCH

THIS IS AN OVERLAY PATCH

### L.4 PATCH ERROR MESSAGES

Error messages which may occur under PATCH follow.

| <u>Message</u>        | <u>Meaning</u>   |
|-----------------------|--|
| ?ADDR NOT IN SEG?     | The address is not in the specified segment.   |
| ?BAD SWITCH?          | Typed a switch other than /O or /M.  |
| ?BOTTOM ADDR WRONG?   | The bottom address specified or contained in location 42 of an overlay file is incorrect. Specify the correct one using the b:B command. |
| ?INCORRECT FILE SPEC? | The response to the "FILE NAME --" message was not of the correct form. Try again.   |
| ?INSUFFICIENT CORE?   | PATCH did not have enough memory to hold the file's device handler plus the internal "segment table." This message should not occur.     |
| ?INVALID RELOC REG?   | Tried to reference a relocation register outside the range 0-7.  |
| ?INVALID SEG NO?      | The segment number S: does not exist.  |
| ?MUST OPEN WORD?      | The @ command was typed when a byte location was open.   |
| ?MUST SPECIFY SEG?    | The address referenced is not in the root section; a segment number S: must be used.   |



## PATCH

|                       |   |
|-----------------------|---|
| ?NO ADDR OPEN?        | The <line feed>, ↑ or @ command was typed when no location was open.                |
| ?NOT IN PROGR BOUNDS? | Tried to open a location beyond the end of the file.                                |
| ?ODD ADDRESS?         | Tried to open a word address which was odd. (Use "\".)                              |
| ?ODD BOTTOM ADDR?     | The bottom address specified or contained in location 42 of an overlay file is odd. |
| ?PROG HAS NO SEGS?    | The file specified as an overlay file is not.                                       |
| ?READ ERROR?          | File I/O error in reading.  |
| ?WRITE ERROR?         | File I/O error in writing.  |



## APPENDIX M

### PATCHO

The RT-11 PATCHO program is used to correct and update object modules (files output by the assemblers or by the FORTRAN compiler). It is particularly useful when making corrections to routines that are in .OBJ format for which the source files are not available. PATCHO cannot be used to patch libraries built by LIBR, but it can be used to patch the OBJ modules from which a library is built.

#### M.1 CALLING AND USING PATCHO

To run PATCHO type the command:

R PATCHO

in response to the dot printed by the Keyboard Monitor. When PATCHO is ready to accept commands, an asterisk is printed. The standard RT-11 command string is not used for PATCHO. Specific commands (described in Section M.2) are typed in response to the asterisk.

Type CTRL C to halt PATCHO at any time and return control to the monitor. To restart PATCHO, type R PATCHO or the REENTER command in response to the monitor's dot.

#### M.2 PATCHO COMMANDS

There are nine commands and arguments accepted by PATCHO. All arguments to a command must be separated from the command name by one or more spaces (e.g., POINT TOP is acceptable, POINTTOP is not). Commands are terminated by a carriage return.

##### M.2.1 OPEN Command

The OPEN command sets input and output file names. When the command is given, PATCHO prints:

ENTER INPUT FILE\*



## PATCHO

on the console terminal, and waits for a standard RT-11 device and file specification to be entered (dev:filnam.ext) . After the input specification is given, followed by a carriage return, PATCHO responds with:

ENTER OUTPUT FILE\*

A device and file specification (followed by a carriage return) for the desired output file is now accepted from the console.

### NOTE

There is no default extension. Therefore, the user must explicitly specify the filename and extension.

Example:

```
*OPEN <CR>
ENTER INPUT FILE *RK1:OTS.OBJ
ENTER OUTPUT FILE *RK1:NEWOTS.OBJ
```

### M.2.2 POINT Command

The POINT command locates an object module of a given name (used with concatenated object modules) and prepares it for subsequent WORD, BYTE, or DUMP operations. POINT takes one argument--the module name to be located.

Example:

```
*POINT OTINIT
```

### M.2.3 WORD Command

The WORD command modifies a given word in an object module. There may be several arguments to the command, entered as illustrated below (the BYTE command is also included since its arguments are exactly the same; refer to Section M.2.4):

|                  | <u>Address Specifier</u> |   | <u>Value Specifier</u> |
|------------------|--------------------------|---|------------------------|
| { WORD<br>BYTE } | CSECT + OFFSET           | = | { # } NAME OF          |
|                  | NAME                     |   | { % } CSECT OR         |
|                  |                          |   | GLOBAL SYMBOL          |
|                  |                          |   | { +<br>- } OFFSET      |

where in the Address Specifier:

1. CSECT is the name of the CSECT which contains the word to be modified. The CSECT must be present in the current module (the one specified in the POINT command).



## PATCHO

The CSECT argument is optional; if omitted, the blank CSECT is assumed. To represent a CSECT name which contains embedded blanks (. ABS.), a backarrow (SHIFT O) or underscore character should be used in place of each embedded blank.

2. OFFSET is the octal location within the CSECT that is to be modified. OFFSET must be present.
3. = is a delimiter and must be present.

In the Value Specifier:

1. If # is used, the absolute value of the OFFSET in the Value Specifier field is placed in the target location.

For example:

```
*WORD PROG+24=#4
```

This command patches location 24 in CSECT PROG to contain the value 4.

2. If % is used, displaced relocation is generated on the Value Specifier. This mode is used for PC-relative references. For example:

```
*WORD PROG+24=%PROG1+24
```

This command patches PROG+24 to contain the value:

Address (PROG1+24)-Address (PROG+26)

3. If neither # or % are specified, the address of the Value Specifier is placed in the target location. For example:

```
*WORD PROG+24=14
```

This causes the word at PROG+24 to contain the address of the word at PROG+14, while:

```
*WORD PROG+24=SYMBOL+0
```

causes the word at PROG+24 to contain the address of the global symbol "SYMBOL".

4. GLOBAL is optional and is the name of a global symbol whose value (address) is to be used in the word to be modified.
5. The characters + or - are optional and indicate the sign of the following OFFSET. Either a + or a - must be present if an OFFSET is indicated.

### M.2.4 BYTE Command

The BYTE command modifies a given byte in an object module. The arguments are the same as for WORD, explained in Section M.2.3.



## PATCHO

### Example:

```
*BYTE CSECTA+21=#101
```

The byte in CSECTA whose offset is 21 is patched to contain octal 101 (ASCII "A").

### M.2.5 DUMP Command

The DUMP command prints the contents of the object module currently being pointed to and causes an automatic POINT to the next module in the input file, if any. Use the monitor SET LP CR command first for correct output format.

#### NOTE

LIST and DUMP go to LP: (lineprinter) by default. If the user wants LIST and DUMP to go to the terminal, he must execute an ASSIGN TT:6 command before typing R PATCHO.

Refer to Section M.4 for an example of the DUMP command.

### M.2.6 LIST Command

The LIST command lists the names of all the object modules in the input file in the order in which they appear in the file. A POINT command should be given after the LIST command to assure that PATCHO is positioned at the desired module for the next operation. LIST lists the names of all object modules in a file without changing the module being pointed to.

Refer to Section M.4 for an example of the LIST command.

### M.2.7 EXIT Command

The EXIT command returns to the operating system terminating a patch session and closing a file. The EXIT command takes no argument. As the EXIT command is executed, ENTER CHECKSUM:- - - is displayed. The user should type a six-digit octal number corresponding to the appropriate checksum for the patch (if any). If the checksum does not match, an error message appears indicating a possible typing error. See the example in Section M.2.8.

### M.2.8 DEC Command

The DEC command is used when the proper checksum for the patch being made is unknown. If a DEC command is issued during a patch session, the EXIT command is automatically modified to display:



## PATCHO

### ENTER CHECKSUM:

followed by the correct checksum for the patch just completed. The checksum computed by PATCHO is derived from all of the commands entered during the session, thereby providing a safeguard against typographical errors. If the DEC command is used, it should be the first command given.

```
*DEC
*EXIT
ENTER CHECKSUM: 25767
```

### M.2.9 HELP Command

The HELP command prints an explanation of PATCHO commands. The HELP command takes no arguments.

### M.3 PATCHO LIMITATIONS

Note the following limitations on the use of PATCHO:

1. PATCHO only works with object modules or concatenated object modules, not with libraries.
2. PATCHO always copies its input file to its output file as it makes changes.
3. As a consequence of 2 (above) the POINT command can only be used to point to modules that physically follow the current module in the input file.
4. If changes are being made to a file, PATCHO must be given an EXIT command when all the changes are complete in order to assure that the entire input is copied to the output (i.e., CTRL C should not be used).
5. The LIST and DUMP commands do not work in an 8K system (12K is required). An ERROR 30 occurs if this is violated.

### M.4 EXAMPLES

The following is an example of the PATCHO command LIST in which a few of the names of object modules in the library file (OTS.OBJ) are listed:

```
*LIST
OBJECT MODULES:
ERRSS
IVEC
IVECP
IPVEC
FVEC
FVECP
```



# PATCHO

PPVEC  
DVEC  
DVECP  
DPVEC  
LVEC  
LVECP  
LPVEC  
ERRS  
ADTS  
OTINIT

The next example is a sample of output produced by the PATCHO DUMP Command. The module is dumped by formatted binary block.

\*DUMP

DUMP OF MODULE LPS0

BLOCK TYPE GSD

| GLOBAL | USAGE    | DEFINED | RELOC | EXTERNAL | SIZE/ADRS |
|--------|----------|---------|-------|----------|-----------|
| LPS0   | MOD NAME | NO      | NO    | NO       | 0         |
| . ABS. | CSECT    | YES     | NO    | NO       | 0         |
| ERRARG | GLOBAL   | NO      | NO    | YES      | 0         |
| ERRPDL | GLOBAL   | NO      | NO    | YES      | 0         |
| ERRSYN | GLOBAL   | NO      | NO    | YES      | 0         |

BLOCK TYPE GSD

| GLOBAL | USAGE  | DEFINED | RELOC | EXTERNAL | SIZE/ADRS |
|--------|--------|---------|-------|----------|-----------|
| EVAL   | GLOBAL | NO      | NO    | YES      | 0         |

.

BLOCK TYPE GSD

| GLOBAL | USAGE    | DEFINED | RELOC | EXTERNAL | SIZE/ADRS |
|--------|----------|---------|-------|----------|-----------|
| USE    | GLOBAL   | YES     | YES   | YES      | 104       |
| . ABS. | TRAN ADR | YES     | NO    | NO       | 1         |

BLOCK TYPE GSD END

BLOCK TYPE RLD

| ADDRESS | RLD TYPE | GLOBAL | OFFSET |
|---------|----------|--------|--------|
|---------|----------|--------|--------|

|      |           |  |   |
|------|-----------|--|---|
| 1772 | LC TR DEF |  | 0 |
|------|-----------|--|---|

BLOCK TYPE TXT

| ADDRESS | CONTENTS |
|---------|----------|
| 0       | 0 0 0 0  |
| 2       | 0 0 0 0  |
| 4       | 0 0 0 0  |
| 6       | 0 0 0 0  |
| 10      | 0 0 0 0  |
| 12      | 0 0 0 0  |

.

|      |       |     |    |
|------|-------|-----|----|
| 1476 | 12602 | 202 | 25 |
| 1500 | 12600 | 200 | 25 |
| 1502 | 207   | 207 | 0  |

BLOCK TYPE MOD END



## PATCHO

### M.5 PATCHO ERROR MESSAGES

| <u>Message</u>                      | <u>Explanation</u>  |
|-------------------------------------|---|
| ?BAD CHECKSUM?                      | A formatted binary block in the input file has a checksum which does not agree with that calculated for its contents.   |
| ?BAD OBJ?                           | The input file contains information which cannot be interpreted as an object module.  |
| ?DUMP ERROR?                        | An input/output error occurred while dumping a module.  |
| ?ILLEGAL COMMAND?                   | A command line was not recognized, or it was not in the proper format for the particular command.   |
| ?BAD PATCH?                         | Checksum entered on exit does not agree with that calculated by PATCHO (the patch was made as specified). This probably indicates a typing error.   |
| ?MODULE NOT FOUND?                  | The module requested in a POINT command was not found in the input file between the position of the file at the time of the point and the end.  |
| ?MORE THAN 15 CHANGES?              | Too many changes have been specified for a particular module. The patch should be broken up into several steps.   |
| ?MORE THAN 5 CSECTS REQUIRE CHANGE? | An attempt has been made to patch locations in too many different CSECTS. The patch should be made in several steps.  |
| ?NO FILE OPEN?                      | An attempt was made to use a command other than "DEC" or "HELP" before an OPEN command was issued.  |
| ?OFFSET?                            | The offset supplied in a WORD or BYTE command is not an octal number, or is in improper format.   |
| ?OUTPUT ERROR?                      | A hardware error (or possibly a write-lock condition) occurred while attempting to write the output file.   |
| ?OUTPUT FILE TOO SMALL?             | The space allocated to the output file is too small. This may be corrected by compressing the device with PIP (if enough total space is free on the device), or by using another device for output. |



## PATCHO

### M.5.1 Run-Time Error Messages

Because PATCHO is a FORTRAN program, run-time error messages may occur. To find a complete explanation of each run-time error refer to the RT-11 FORTRAN COMPILER AND OBJECT TIME SYSTEM USER'S MANUAL (DEC-11-LRFPA-A-D). Listed below are four of the most important run-time error messages which may be encountered.

- |    |       |  |
|----|-------|--|
| 23 | FATAL | HARDWARE I/O ERROR<br>A hardware error has been detected during an I/O operation.                            |
| 28 | FATAL | OPEN FAILED FOR FILE<br>A file could not be found.   |
| 29 | FATAL | NO ROOM FOR DEVICE HANDLER<br>There is not enough free memory left to accommodate a specific device handler. |
| 30 | FATAL | NO ROOM FOR BUFFERS<br>There is not enough free memory left to set up required I/O buffers.                  |



## APPENDIX N

### DISPLAY FILE HANDLER

This appendix describes the assembly language graphics support provided under RT-11 for the GT40, GT44, and DECLab-11 display hardware systems.

The following manuals are suggested for additional reference:

For GT40 users:

1. GT40 USER'S GUIDE (DEC-11-HGTGA-A-D)

For GT44 users:

1. GT44 USER'S GUIDE (DEC-11-HGT44-A-D)

For DECLab-11 users:

1. VT-11 GRAPHICS DISPLAY PROCESSOR MANUAL (DEC-11-HVGTA-A-D)
2. RT-11 BASIC REFERENCE MANUAL (DEC-11-LBACA-B-D)

#### N.1 DESCRIPTION

The GT40, GT44, and DECLab-11 have hardware configurations that include a display processor and CRT (cathode ray tube) display. The GT44 has a 17-inch tube; the GT40 and DECLab-11 use a 12-inch tube. Both systems are equipped with light pens and hardware character and vector generators, and are capable of high-quality graphics. The Display File Handler supports this graphics hardware at the assembly language level under the RT-11 monitor.

##### N.1.1 Assembly Language Display Support

The Display File Handler is not an RT-11 device handler, since it does not use the I/O structure of the RT-11 monitor. For example, it is not possible to use PIP to transfer a text file to the display via the Display File Handler. Rather, the Display File Handler provides the graphics programmer the means for the display of graphics files and the easy management of the display processor. Included in its



## Display File Handler

capabilities are such services as interrupt handling, light pen support, tracking object, and starting and stopping of the display processor.

The Display File Handler manages the display processor by means of a base segment (called VTBASE) which contains interrupt handlers, an internal display file and some pointers and flags. The display processor cycles through the internal display file; any user graphics files to be displayed are accessed via display subroutine calls from the Handler's display file. In this way, the Display File Handler exerts control over the display processor, relieving the assembly language user of the task.

Through the Display File Handler, the programmer can insert and remove calls to display files from the Handler's internal display file. Up to two user files may be inserted at one time, and that number may be increased by re-assembling the Handler. Any user file inserted for display may be blanked (the subroutine call to it bypassed) and unblanked by macro calls to the Display File Handler.

Since the Handler treats all user display files as graphics subroutines to its internal display file, a display processor subroutine call is required. This is implemented with software, using the display stop instruction, and is available for user programs. This instruction and several other extended instructions implemented with the display stop instruction are described in Section N.3.

The facilities of the Display File Handler are accessed through a library of macros (VTMAC) which generate calls to a set of subroutines in VTLIB. VTMAC is the MACRO library, and its call protocol is similar to that of the RT-11 macros. The expansion of the macros is shown in Section N.6. VTMAC also contains, for convenience in programming, the set of recommended display processor instruction mnemonics and their values. The mnemonics are listed in Section N.7 and are used in the examples throughout this appendix.

VTCALL through VTCAL4 are the set of subroutines which service the VTMAC calls. They include functions for display file and display processor management. These are described in detail in Section N.2. VTCALL through VTCAL4 are currently constructed, along with the base segment VTBASE, as a library of (five) modules, called VLIB.

### N.1.2 Monitor Display Support

The RT-11 monitor, under Version 2, directly supports the display as console device. A keyboard monitor command, GT ON (GT OFF) described in Section 2.7.1, permits the selection of the display as console device. Selection results in the allocation of approximately 1.25K words of memory for text buffer and code. The buffer holds approximately 2000 characters.

The text display includes a blinking cursor to indicate the position in the text where a character is added. The cursor initially appears at the top left corner of the text area. As lines are added to the text the cursor moves down the screen. When the maximum number of lines are on the screen, the top line is deleted from the text buffer when the line feed terminating a new line is received. This causes the appearance of "scrolling", as the text disappears off the top of the display.



## Display File Handler

When the maximum number of characters have been inserted in the text buffer, the scroller logic deletes a line from the top of the screen to make room for additional characters. Text may appear to move (scroll) off the top of the screen while the cursor is in the middle of a line.

The Display File Handler can operate simultaneously with the scroller program, permitting graphic displays and monitor dialogue to appear on the screen at the same time. It does this by inserting its internal display file into the display processor loop through the text buffer.

Four scroller control characters provide the user with the capability of halting the scroller, advancing the scrolling in page sections, and printing hard copy from the scroller. These are described in Chapter 2.

### NOTE

The scroller logic does not limit the length of a line, but the length of text lines affects the number of lines which may be displayed, since the text buffer is finite. As text lines become longer, the scroller logic may delete extra lines to make room for new text, temporarily decreasing the number of lines displayed.

## N.2 DESCRIPTION OF GRAPHICS MACROS

The facilities of the Display File Handler are accessed through a set of macros, contained in VTMAC, which generate assembly language calls to the Handler at assembly time. The calls take the form of subroutine calls to the subroutines in VTLIB. Arguments are passed to the subroutines through register 0 and, in the case of the .TRACK call, through both register 0 and the stack.

This call convention is similar to Version 1 RT-11 I/O macro calls, except that the subroutine call instruction is used instead of the EMT instruction. If a macro requires an argument but none is specified, it is assumed that the address of the argument has already been placed in register 0. The programmer should not assume that R0 is preserved through the call.

### N.2.1 .BLANK

The .BLANK request temporarily blanks the user display file specified in the request. It does this by by-passing the call to the user display file, which prevents the display processor from cycling through the user file, effectively blanking it. This effect can later



## Display File Handler

be cancelled by the .RESTR request, which restores the user file. When the call returns, the user is assured the display processor is not in the file that was blanked.

Macro Call: .BLANK .faddr

where: .faddr is the address of the user display file to be blanked.

### Errors:

No error is returned. If the file specified was not found in the Handler file or has already been blanked, the request is ignored.

## N.2.2 .CLEAR

The .CLEAR request initializes the Display File Handler, clearing out any calls to user display files and resetting all of the internal flags and pointers.

After initialization with .LNKRT (Section N.2.4), the .CLEAR request can be used any time in a program to clear the display and to reset pointers. All calls to user files are deleted and all pointers to status buffers are reset. They must be re-inserted if they are to be used again.

Macro Call: .CLEAR

### Errors:

None.

### Example:

This example uses a .CLEAR request to initialize the Handler, then later uses the .CLEAR to re-initialize the display. The first .CLEAR is used for the case when a program may be restarted after a CTRL C or other exit.

```

      BR      RSTRT
EX11  BIS     #20000,0#44      ;SET RENTER BIT IN JSW
RSTRT: .UNLNK                    ;CLEARS LINK FLAG FOR RESTART
      .LNKRT                    ;SET UP VECTORS, START DISPLAY
      .CLEAR                    ;INITIALIZE HANDLER.
      .INSRT #FILE1             ;DISPLAY A PICTURE
131   .TTYIN                    ;WAIT FOR A KEY STRIKE
      CMPB    #12,R0            ;LINE FEED?
      BNE     13                ;NO, LOOP
      .CLEAR                    ;YES, CLEAR DISPLAY
      .INSRT #FILE2             ;DISPLAY NEW PICTURE
      .
      .
FILE11 POINT                      ;AT POINT (0,500)
      0
      500
      LONGV                      ;DRAW A LINE
      500,INTX                  ;TO (500,500)
      0
      ORET
      0
```



## Display File Handler

```
FILE2:  POINT          JAT POINT (500,0)
        500
        0
        LONGV          JDRAW A LINE
        0!INTX         JTO (500,500)
        500
        ORET
        0

        .END          EX1
```

### N.2.3 .INSRT

The .INSRT request inserts a call to the user display file specified in the request into the Display File Handler's internal display file. .INSRT causes the display processor to cycle through the user file as a subroutine to the internal file. The handler permits two user files at one time. The call inserted in the handler looks like the following:

```
DJSR      ;DISPLAY SUBROUTINE
.+4       ;RETURN ADDRESS
.faddr    ;SUBROUTINE ADDRESS
```

The call to the user file is removed by replacing its address with the address of a null display file. The user file is blanked by replacing the DJSR with a DJMP instruction, bypassing the user file.

Macro Call: .INSRT .faddr

where: .faddr is the address of the user display file to be inserted.

### Errors:

The .INSRT request returns with the C bit set if there was an error in processing the request. An error occurs only when the Handler's display file is full and cannot accept another file. If the user file specified exists, the request is not processed. Two display files with the same starting address cannot be inserted.

### Example:

See the examples in Sections N.2.2 and N.2.4.

### N.2.4 .LNKRT

The .LNKRT request sets up the display interrupt vectors and possibly links the Display File Handler to the scroll text buffer in the RT-11 monitor. It must be the first call to the Handler, and is used whether or not the RT-11 monitor is using the display for console output (i.e., the KMON command GT ON has been entered).

The .LNKRT request used with the Version 2 RT-11 monitor enables a display application program to determine the environment in which it is operating. Error codes are provided for the situations where there is no display hardware present on the system or the display hardware is already being used by another task (e.g., a foreground job in the Foreground/Background version).



## Display File Handler

The existence of the monitor scroller and the size of the Handler's subpicture stack are also returned to the caller. If a previous call to .LNKRT was made without a subsequent .UNLNK, the .LNKRT call is ignored and an error code is returned.

Macro Call: .LNKRT

### Errors:

Error codes are returned in R0, with the N condition bit set.

| <u>Code</u> | <u>Meaning</u>                                      |
|-------------|---|
| -1          | No VT11 display hardware is present on this system. |
| -2          | VT11 hardware is presently in use.                  |
| -3          | Handler has already been linked.                    |

On completion of a successful .LNKRT request, R0 will contain the display subroutine stack size, indicating the depth to which display subroutines may be nested. The N bit will be zero.

If the RT-11 monitor scroll text buffer was not in memory at the time of the .LNKRT, the C bit will be returned set. The KMON commands GT ON and GT OFF cannot be issued while a task is using the display.

### Example:

```

START: .LNKRT          ;LINK TO MONITOR
      BHI  ERROR      ;ERROR DOING LINK
      BCS  CONT       ;NO SCROLL IF C SET
      .SCROL #SBUF     ;ADJUST SCROLL PARAMETERS
CONT:  .INSRT #FILE1   ;DISPLAY A PICTURE
IS:    .TTYIN         ;WAIT FOR KEY STRIKE
      CMPB #12,R0     ;LINE FEED?
      BNE  IS         ;NO, LOOP
      .UNLNK         ;YES, UNLINK AND EXIT
      .EXIT

SBUF:  .BYTE 5        ;LINE COUNT OF 5
      .BYTE 7        ;INTENSITY 7(SCALE OF 1-8)
      .WORD 1000      ;POSITION OF TOP LINE

FILE1: POINT          ;AT POINT (500,500)
      500
      500
      CHAR           ;DISPLAY SOME TEXT
      .ASCII /THIS IS FILE1. TYPE CR TO EXIT/
      .EVEN
      DRET
      0

```

ERROR: Error routine



## Display File Handler

### N.2.5 .LPEN

The .LPEN request transfers the address of a light pen status data buffer to VTBASE. Once the buffer pointer has been passed to the Handler, the light pen interrupt handler in VTBASE will transfer display processor status data to the buffer, depending on the state of the buffer flag.

The buffer must have seven contiguous words of storage. The first word is the buffer flag, and it is initially cleared (set to zero) by the .LPEN request. When a light pen interrupt occurs, the interrupt handler transfers status data to the buffer and then sets the buffer flag non-zero. The program can loop on the buffer flag when waiting for a light pen hit (although doing this will tie up the processor, and in a Foreground/Background environment, timed waits would be more desirable). No further data transfers take place, despite the occurrence of numerous light pen interrupts, until the buffer flag is again cleared to zero. This permits the program to process the data before it is destroyed by another interrupt.

The buffer structure looks like this:

```
Buffer Flag
Name
Subpicture Tag
Display Program Counter (DPC)
Display Status Register (DSR)
X Status Register (XSR)
Y Status Register (YSR)
```

The Name value is the contents of the software Name Register (described in N.3.5) at the time of interrupt. The Tag value is the tag of the subpicture being displayed at the time of interrupt. The last four data items are the contents of the display processor status registers at the time of interrupt. They are described in detail in Table N-1.

Macro Call: .LPEN .baddr

where: .baddr is the address of the 7-word light pen status data buffer.

Errors:

None.

If a .LPEN was already issued and a buffer specified, the new buffer address replaces the previous buffer address. Only one light pen buffer can be in use at a time.

Example:

```
      .INSRT  #LFILE      ;DISPLAY LFILE
      .LPEN   #LBUF       ;SET UP LPEN BUFFER
LOOP:  TST    LBUF        ;TEST LBUF FLAG, WHICH
      BEQ     LOOP       ;WILL BE SET NON-ZERO
                               ;ON LIGHT PEN HIT.

      ; PROCESS DATA IN LBUF HERE,
      CLR     LBUF       ;CLEAR THE BUFFER FLAG,
                               ;PERMITTING ANOTHER "HIT".
      BR      LOOP       ;GO WAIT FOR IT,
```



# Display File Handler

LBUF1 .BLKW 7  
LFILE1

ISEVEN WORD LPEN BUFFER

•  
•  
•

Table N-1  
Description of Display Status Words

| Bits  | Significance   |
|---|--|
| <u>DISPLAY PROGRAM COUNTER (DPC=172000)</u> |  |
| 0-15  | Address of display processor program counter at time of interrupt. |
| <u>DISPLAY STATUS REGISTER (DSR=172002)</u> |  |
| 0-1   | Line Type  |
| 2   | Spare  |
| 3   | Blink  |
| 4   | Italics  |
| 5   | Edge Indicator   |
| 6   | Shift Out  |
| 7   | Light Pen Flag   |
| 8-10  | Intensity  |
| 11-14                                       | Mode   |
| 15  | Stop Flag  |
| <u>X STATUS REGISTER (XSR=172004)</u>       |  |
| 0-9   | X Position   |
| 10-15                                       | Graphplot Increment  |
| <u>Y STATUS REGISTER (YSR=172006)</u>       |  |
| 0-9   | Y Position   |
| 10-15                                       | Character Register   |



## Display File Handler

### N.2.6 .NAME

The .NAME request has been added to the Version 2 Handler. The contents of the name register are now stacked when a subpicture call is made. When a light pen interrupt occurs, the contents of the name register stack may be recovered if the user program has supplied the address of a buffer through the .NAME request.

The buffer must have a size equal to the stack depth (default is 10) plus one word for the flag. When the .NAME request is entered, the address of the buffer is passed to the Handler and the first word (the flag word) is cleared. When a light pen hit occurs, the stack's contents are transferred and the flag is set non-zero.

Macro Call: .NAME .baddr

where: .baddr is the address of the name register buffer.

#### Errors:

None.

If a .NAME request has been previously issued, the new buffer address replaces the previous buffer address.

### N.2.7 .REMOV

The .REMOV request removes the call to a user display file previously inserted in the handler's display file by the .INSRT request. All reference to the user file is removed, unlike the .BLANK request, which merely bypasses the call while leaving it intact.

Macro Call: .REMOV .faddr

where: .faddr is the address of the display file to be removed.

#### Errors:

No errors are returned. If the file address given cannot be found, the request is ignored.

### N.2.8 .RESTR

The .RESTR request restores to view a user display file that was previously blanked by a .BLANK request. It removes the by-pass of the call to the user file, so that the display processor once again cycles through the user file.

Macro Call: .RESTR .faddr

where: .faddr is the address of the user file that is to be restored to view.

#### Errors:

No errors are returned. If the file specified cannot be found, the request is ignored.



## Display File Handler

### N.2.9 .SCROL

This request is used to modify the appearance of the Display Monitor's text display. The .SCROL request permits the programmer to change the maximum line count, intensity and the position of the top line of text of the scroller. The request passes the address of a two-word buffer which contains the parameter specifications. The first byte is the line count, the second byte is the intensity, and the second word is the Y position. Both line count and Y position must be octal numbers. The intensity may be a number from 1 to 8, ranging from lowest to highest intensity. If an intensity of 1 is specified, the scroller text will be almost unnoticeable at a BRIGHTNESS knob setting less than one-half. The scroller parameter change is temporary, since an .UNLNK or CTRL C restores the previous values.

Macro Call: .SCROL .baddr

where:      baddr                      is the address of the two-word  
   scroll parameters buffer.

#### Errors:

No errors are returned. No checking is done on the values of the parameters. A zero argument is interpreted to mean that the parameter value is not to be changed. A negative argument causes the default parameter value to be restored.

#### Example:

```
                .SCROL  #SCBUF          ;ADJUST SCROLL PARAMETERS
                :
                :
SCBUF:  .BYTE    5                      ;DECREASE # LINES TO 5,
        .BYTE    0                      ;LEAVE INTENSITY UNCHANGED,
        .WORD    300                    ;TOP LINE AT Y = 300.
```

### N.2.10 .START

The .START request starts the display processor if it was stopped by a .STOP directive. If the display processor is running, it is stopped first, then restarted. In either case, the subpicture stack is cleared and the display processor is started at the top of the handler's internal display file.

Macro Call: .START

#### Errors:

None.

### N.2.11 .STAT

The .STAT request transfers the address of a seven-word status buffer to the display stop interrupt routine in VTBASE. Once the transfer has been made, display processor status data is transferred to the buffer by the display stop interrupt routine in VTBASE whenever a



## Display File Handler

.DSTAT or .DHALT instruction is encountered (see Sections N.3.3 and N.3.4). The transfer is made only when the buffer flag is clear (zero). After the transfer is made, the buffer flag is set non-zero and the .DSTAT or .DHALT instruction is replaced by a .DNOP (Display NOP) instruction.

The status buffer must be a seven-word, contiguous block of memory. Its contents are the same as the light pen status buffer. For a detailed description of the buffer and an explanation of the status words, see section N.2.5 and Table N-1.

Macro Call: .STAT .baddr

where: .baddr is the address of the status  
buffer receiving the data.

### Errors:

No errors are indicated. If a buffer was previously set up, the new buffer address is replaced as the old buffer address.

## N.2.12 .STOP

The .STOP request "stops" the display processor. It actually effects a stop by preventing the DPU from cycling through any user display files. It is useful for stopping the display during modification of a display file, a risky task when the display processor is running. However, a .BLANK could be equally useful for this purpose, since the .BLANK request does not return until the display processor has been removed from the user display file being blanked.

Macro Call: .STOP

### Errors:

None.

### NOTE

Since the display processor must cycle through the text buffer in the Display Monitor in order for console output to be processed, the text buffer remains visible after a .STOP request is processed, but all user files disappear.

## N.2.13 .SYNC/.NOSYN

The .SYNC and .NOSYN requests provide program access to the power line synchronization feature of the display processor. The .SYNC request enables synchronization and the .NOSYN request disables it (the default case).

Synchronization is achieved by stopping the display and restarting it when the power line frequency reaches a trigger point, e.g., a peak or zero-crossing. Synchronization has the effect of fixing the display



## Display File Handler

refresh time. This may be useful in some cases where small amounts of material are displayed but the amount frequently changes, causing changes in intensity. In most cases, however, using synchronization increases flicker.

Macro Calls: .SYNC  
.NOSYN

Errors:

None.

### N.2.14 .TRACK

The .TRACK request causes the tracking object to appear on the display CRT at the position specified in the request. The tracking object is a diamond-shaped display figure which is light-pen sensitive. If the light pen is placed over the tracking object and then moved, the tracking object follows the light pen, trying to center itself on the pen.

The tracking object first appears at a position specified in a two-word buffer whose address was supplied with the .TRACK request. As the tracking object moves to keep centered on the light pen, the new center position is returned to the buffer. A new set of X and Y values is returned for each light pen interrupt.

The tracking object cannot be lost by moving it off the visible portion of the display CRT. When the edge flag is set, indicating a side of the tracking object is crossing the edge of the display area, the tracking object stops until moved toward the center. To remove the tracking object from the screen, repeat the .TRACK request without arguments.

The .TRACK request may also include the address of a completion routine as the second argument. If a .TRACK completion routine is specified, the light pen interrupt handler passes control to the completion routine at interrupt level. The completion routine is called as a subroutine and the exit statement must be an RTS PC. The completion routine must also preserve any registers it may use.

Macro Call: .TRACK .baddr, .croutine

|        |           |   |
|--------|-----------|---|
| where: | .baddr    | is the address of the two-word buffer containing the X and Y position for the track object. |
|        | .croutine | is the address of the completion routine.   |

Errors:

None.

Example:

See Section N.10.



## Display File Handler

### N.2.15 .UNLNK

The .UNLNK request is used before exiting from a program. In the case where the scroller is present, .UNLNK breaks the link, established by .LNKRT, between the Display File Handler's internal display file and the scroll file in the Display Monitor. The display processor is started cycling in the scroll text buffer, and no further graphics may be done until the link is established again. In the case where no scroller exists, the display processor is simply left stopped.

Macro Call: .UNLNK

#### Errors:

No errors are returned. An internal link flag is checked to determine if the link exists. If it does not exist, the request is ignored.

## N.3 EXTENDED DISPLAY INSTRUCTIONS

The Display File Handler offers the assembly language graphics programmer an extended display processor instruction set, implemented in software through the use of the Load Status Register A (LSRA) instruction. The extended instruction set includes: subroutine call, subroutine return, display status return, display halt, and load name register.

### N.3.1 DJSR Subroutine Call Instruction

The DJSR instruction (octal code is 173400) simulates a display subroutine call instruction by using the display stop instruction (LSRA instruction with interrupt bits set). The display stop interrupt handler interprets the non-zero word following the DJSR as the subroutine return address, and the second word following the DJSR as the address of the subroutine to be called. The instruction sequence is:

```
DJSR \
Return address
Subroutine address
```

#### Example:

To call a subroutine SQUARE:

```
POINT          ;POSITION BEAM
100             ;AT (100,100)
100
DJSR            ;THEN CALL SUBROUTINE
.+4
SQUARE          ;TO DRAW A SQUARE
DRET
0
```

The use of the return address preceding the subroutine address offers several advantages. BASIC/GT uses the return address to branch around subpicture tag data stored following the subpicture address. This structure is described in Section N.5.3. In addition, a subroutine may be temporarily bypassed by replacing the DJSR code with a DJMP instruction, without the need to stop the display processor to make the by-pass.



## Display File Handler

The address of the return address is stacked by the display stop interrupt handler on an internal subpicture stack. The stack depth is conditionalized and has a default depth of 10. If the stack bottom is reached, the display stop interrupt handler attempts to protect the system by rejecting additional subroutine calls. In that case, the portions of the display exceeding the legal stack depth will not be displayed.

### N.3.2 DRET Subroutine Return Instruction

The DRET instruction provides the means for returning from a display file subroutine. It uses the same octal code as DJSR, but with a single argument of zero. The DRET instruction causes the display stop interrupt handler to pop its subpicture stack and fetch the subroutine return address.

Example:

```
SQUARE: LONGV                                ;DRAW A SQUARE
        100;INTX
        0
        0;INTX
        100
        100;INTX;MINUSX
        0
        0;INTX
        100;MINUSX
        DRET                                ;RETURN FROM SUBPICTURE
        0
```

### N.3.3 DSTAT Display Status Instruction

The DSTAT instruction (octal code is 173420) uses the LSRA instruction to produce a display stop interrupt, causing the display stop interrupt handler to return display status data to a seven-word user status buffer. The status buffer must first have been set up with a .STAT macro call (if not, the DSTAT is ignored and the display is resumed). The first word of the buffer is set non-zero to indicate the transfer has taken place, and the DSTAT is replaced with a DNOP (display NOP). The first word is the buffer flag and the next six words contain name register contents, current subpicture tag, display program counter, display status register, display X register, and display Y register. After transfer of status data, the display is resumed.

### N.3.4 DHALT Display Halt Instruction

The DHALT instruction (octal code is 173500) operates similarly to the DSTAT instruction. The difference between the two instructions is that the DHALT instruction leaves the display processor stopped when exiting from the interrupt. A status data transfer takes place provided the buffer was initialized with a .STAT call. If not, the DHALT is ignored.



## Display File Handler

Example:

```

        .STAT    #SBUF          ;INIT BUFFER
        MOV      #DHALT,STPLOC  ;INSERT DHALT
        .INSRT   #DFILE         ;DISPLAY THE PICTURE
15:     TST       SBUF          ;DHALT PROCESSED?
        BEQ      15            ;NO, WAIT
        .
        SBUF:    .BLKW          7          ;STATUS BUFFER
        OFILE:   POINT         ;POSITION NEAR TOP OF 12" TL
                .WORD          500,1350
                LONGV
                .WORD          0,400
        STPLOC:  DNOP
                DRET
                0

```

### N.3.5 DNAME Load Name Register Instruction

The Display File Handler provides a name register capability through the use of the display stop interrupt. When a DNAME instruction (octal code is 173520) is encountered, a display stop interrupt is generated. The display stop handler stores the argument following the DNAME instruction in an internal software register called the "name register". The current name register contents are returned whenever a DSTAT or DHALT is encountered, and more importantly, whenever a light pen interrupt occurs. The use of a "name" (with a valid range from 1 to 77777) enables the programmer to label each element of the display file with a unique name, permitting the easy identification of the particular display element selected by the light pen.

The name register contents are stacked on a subpicture call and restored on return from the subpicture.

Example:

The SQUARE subroutine with "named" sides:

```

SQUARE: DNAME          ;NAME IS
        10             ;10
        LONGV          ;DRAW A SIDE
        100;INTX
        0
        DNAME          ;THIS SIDE IS NAMED
        11             ;11
        0;INTX         ;STILL IN LONG VECTOR MODE
        100
        DNAME
        12
        100;INTX;MINUSX
        0
        DNAME

```



## Display File Handler

```
13
0:INTX
100:MINUSX
DRET                                ;RETURN FROM SUBPICTURE
0
```

### N.4 USING THE DISPLAY FILE HANDLER

Graphics programs which intend to use the Display File Handler for display processor management can be written in MACRO assembly language. The display code portions of the program may use the mnemonics described in Section N.7. Calls to the Handler should have the format described in Section N.6.

The Display File Handler is supplied in two pieces, a library of MACRO calls and a library containing the Display File Handler modules.

|                       |                             |
|-----------------------|-----------------------------|
| MACRO Library:        | VTMAC.MAC                   |
| Display File Handler: | VTHDLR.OBJ (consisting of:) |
|                       | VTBASE.OBJ                  |
|                       | VTCAL1.OBJ                  |
|                       | VTCAL2.OBJ                  |
|                       | VTCAL3.OBJ                  |
|                       | VTCAL4.OBJ                  |

#### N.4.1 Assembling Graphics Programs

To assemble a graphics program using the display processor mnemonics or the Display Handler macro calls, the file VTMAC.MAC must be assembled with the program, and must precede the program in the assembler command string.

##### Example:

Assume PICTUR.MAC is a user graphics program to be assembled. An assembler command string would look like this:

```
.R MACRO
*PICTUR=VTMAC,PICTUR
```

#### N.4.2 Linking Graphics Programs

Once assembled with VTMAC, the graphics program must be linked with the Display File Handler, which is supplied as a single concatenated object module, VTHDLR.OBJ. The Handler may optionally be built as a library, following the directions in N.8.5. The advantage of using the library when linking is that the Linker will select from the library only those modules actually used. Linking with VTHDLR.OBJ results in all modules being included in the link.

To link a user program called PICTUR.OBJ using the concatenated object module supplied with RT-11:

```
.R LINK
*PICTUR=PICTUR,VTHDLR
*
```



## Display File Handler

To link a program called PICTUR.OBJ using the VTLIB library built by following the directions in N.8.5, be sure to use the Version 2 Linker:

```
.R LINK
*PICTUR=PICTUR,VTLIB
*
```

### VTLIB (Handler Modules):

| <u>Module</u> | <u>CSECT</u> | <u>Contains</u>   |
|---------------|--------------|---|
| VTCAL1        | \$GT1        | .CLEAR<br>.START<br>.STOP<br>.INSRT<br>.REMOV   |
| VTCAL2        | \$GT2        | .BLANK<br>.RESTR  |
| VTCAL3        | \$GT3        | .LPEN<br>.NAME<br>.STAT<br>.SYNC<br>.NOSYN<br>.TRACK                                      |
| VTCAL4        | \$GT4        | .LNKRT<br>.UNLNK<br>.SCROL  |
| VTBASE        | \$GTB        | Memory resident base module<br>containing interrupt handlers<br>and internal display file |

To link a display program using overlays, the modules must be specified individually. The user must acquire the sources and follow the assembly directions in Section N.8. The modules VTCAL1 and VTCAL2 must be simultaneously resident. For example:

```
.R LINK
*PICTUR=PICTUR,VTBASE/C
*VTCAL1,VTCAL2,VTCAL3/O:1/C
*VTCAL4/O:1
```

To link a display program to run as a foreground task:

```
.R LINK
*PICTUR=PICTUR,VTLIB/R
```

## N.5 DISPLAY FILE STRUCTURE

The Display File Handler supports a variety of display file structures, takes over the job of display processor management for the programmer, and may be used for both assembly language graphics programming and for systems program development. For example, the



## Display File Handler

Handler supports the tagged subpicture file structure used by BASIC/GT, as well as simple file structures. These are discussed in this section.

### N.5.1 Subroutine Calls

A subroutine call instruction, with the mnemonic DJSR, is implemented using the display stop (DSTOP) instruction with an interrupt. The display stop interrupt routine in the Display File Handler simulates the DJSR instruction, and this allows great flexibility in choosing the characteristics of the DJSR instruction.

The DJSR instruction stops the display processor and requests an interrupt. The DJSR instruction may be followed by two or more words, and in this implementation the exact number may be varied by the programmer at any time. The basic subroutine call has this form:

```
DJSR
Return Address
Subroutine Address
```

In practice, simple calls to subroutines could look like:

```
DJSR
.WORD      .+4
.WORD      SUB
```

where SUB is the address of the subroutine. Control will return to the display instruction following the last word of the subroutine call. This structure permits a call to the subroutine to be easily by-passed without stopping the display processor, by replacing the DJSR with a display jump (DJMP) instruction:

```
DJMP
.WORD      .+4
.WORD      SUB
```

A more complex display file structure is possible if the return address is generalized:

```
.DJSR
.WORD      NEXT
.WORD      SUB
```

where NEXT is the generalized return address. This is equivalent to the sequence:

```
DJSR
.WORD      .+4
.WORD      SUB
DJMP
.WORD      NEXT
```

It is also possible to store non-graphic data such as tags and pointers in the subroutine call sequence, such as is done in the tagged subpicture display file structure of BASIC/GT. This technique looks like:



## Display File Handler

```
        DJSR
        .WORD      NEXT
        .WORD      SUB
        DATA
NEXT:    .
        .
        .
```

For simple applications where the flexibility of the DJSR instruction described above is not needed and the resultant overhead not desired, the Display File Handler (VTBASE.MAC and VTCALL.MAC) can be conditionally re-assembled to produce a simple DJSR call. If NOTAG is defined during the assembly, the Handler will be configured to support this simple DJSR call:

```
        DJSR
        .WORD      SUB
```

where SUB is the address of the subroutine. Defining NOTAG will eliminate the subpicture tag capability, and with it the tracking object, which uses the tag feature to identify itself to the light pen interrupt handler.

Whatever the DJSR format used, all subroutines and the user main file must be terminated with a subroutine return instruction. This is implemented as a display stop instruction (given the mnemonic DRET) with an argument of zero. A subroutine then has the form:

```
SUB: Display Code
    DRET
    .WORD 0
```

### N.5.2 Main File/Subroutine Structure

A common method of structuring display files is to have a main file which calls a series of display subroutines. Each subroutine will produce a picture element and may be called many times to build up a picture, producing economy of code. If the following macros are defined:

```
.MACRO    CALL <ARG>
DJSR
.WORD      .+4
.WORD      ARG
.ENDM
.MACRO    RETURN
DRET
.WORD      0
.ENDM
```

then a main file/subroutine file structure would look like:

```
;MAIN DISPLAY FILE
;
MAIN:    Display Code
        CALL SUB1      ;CALL SUBROUTINE 1
        Display Code
        CALL SUB2      ;CALL SUBROUTINE 2
                        ;ETC
```



### N.5.3 BASIC/GT Subroutine Structure

The subpicture is constructed as a subroutine call followed by the subroutine. — It is essentially a merger of the main file/subroutine structure into an in-line sequence of calls and subroutines. As such, it facilitates the construction of display files in real time, one of the important advantages of BASIC/GT.

```

SUB1:  DJSR          ;START OF SUBPICTURE 1
        .WORD      SUB2          ;NEXT SUBPICTURE
        .WORD      SUB1+12       ;JUMP TO THIS SUBPICTURE
        .WORD      1             ;TAG = 1
        .WORD      SUB2+6        ;POINTER TO NEXT TAG

```

```

DRET                                RETURN FROM
0                                  SUBPICTURE 1

```

```

1 BODY OF SUBPICTURE 2
  DRET
  .WORD 0
                                ;RETURN FROM
                                ;SUBPICTURE 2

```



JSR %↑07, \$VRMOV



## Display File Handler

|        |   |                             |   |
|--------|---|-----------------------------|---|
| .RESTR | Unblanks the user display file.   | .RESTR .faddr               | .GLOBL \$VRSTR<br>IF NB, .faddr<br>MOV .faddr, %t00<br>.ENDC<br>JSR %t07, \$VRSTR   |
| .SCROL | Adjusts monitor scroller parameters.  | .SCROL .baddr               | .GLOBL \$VSCRL<br>.IF NB, .baddr<br>MOV .baddr, %t00<br>.ENDC<br>JSR %t07, \$VSCRL  |
| .START | Starts the display.   | .START                      | .GLOBL \$VSTRT<br>JSR %t07, \$VSTRT   |
| .STAT  | Sets up status buffer.  | .STAT .baddr                | .GLOBL \$VSTPM<br>.IF NB, .baddr<br>MOV .baddr, %t00<br>.ENDC<br>JSR %t07, \$VSTPM  |
| .STOP  | Stops the display.  | .STOP                       | .GLOBL \$VSTOP<br>JSR %t07, \$VSTOP   |
| .SYNC  | Enables power line sync.  | .SYNC                       | .GLOBL \$SYNC<br>JSR %t07, \$SYNC   |
| .TRACK | Enables the track object.   | .TRACK .baddr,<br>.croutine | .GLOBL \$VTRAK<br>.IF NB, .baddr<br>MOV .baddr, %t00<br>.ENDC<br>.IF NB, .croutine<br>MOV .croutine, -<br>(%t06)<br>.IFF<br>CLR-(%t06)<br>.ENDC<br>.NARG T<br>.IF EQ, T<br>CLR %t00<br>.ENDC<br>JSR %t07, \$VTRAK |
| .UNLNK | Unlinks display handler from RT-11 if linked, otherwise leaves display stopped. | .UNLNK                      | .GLOBL \$VUNLK<br>JSR %t07, \$VUNLK   |

### NOTE 1

|           |  |
|-----------|--|
| .baddr    | Address of data buffer.                |
| .faddr    | Address of start of user display file. |
| .croutine | Address of .TRACK completion routine.  |



## Display File Handler

### NOTE 2

The lines preceded by a dot will not be assembled. The code they enclose may or may not be assembled depending on the conditionals.

### N.7 DISPLAY PROCESSOR MNEMONICS

| <u>Mnemonic</u> | <u>=</u> | <u>Value</u> | <u>Function</u>              |
|-----------------|----------|--------------|------------------------------|
| CHAR            | =        | 100000       | Character Mode               |
| SHORTV          | =        | 104000       | Short Vector Mode            |
| LONGV           | =        | 110000       | Long Vector Mode             |
| POINT           | =        | 114000       | Point Mode                   |
| GRAPHX          | =        | 120000       | Graphplot X Mode             |
| GRAPHY          | =        | 124000       | Graphplot Y Mode             |
| RELATV          | =        | 130000       | Relative Point Mode          |
| INT0            | =        | 2000         | Intensity 0<br>(Dimmest)     |
| INT1            | =        | 2200         | Intensity 1                  |
| INT2            | =        | 2400         | Intensity 2                  |
| INT3            | =        | 2600         | Intensity 3                  |
| INT4            | =        | 3000         | Intensity 4                  |
| INT5            | =        | 3200         | Intensity 5                  |
| INT6            | =        | 3400         | Intensity 6                  |
| INT7            | =        | 3600         | Intensity 7<br>(Brightest)   |
| LPOFF           | =        | 100          | Light Pen Off                |
| LPON            | =        | 140          | Light Pen On                 |
| BLKOFF          | =        | 20           | Blink Off                    |
| BLKON           | =        | 30           | Blink On                     |
| LINE0           | =        | 4            | Solid Line                   |
| LINE1           | =        | 5            | Long Dash                    |
| LINE2           | =        | 6            | Short Dash                   |
| LINE3           | =        | 7            | Dot Dash                     |
| DJMP            | =        | 160000       | Display Jump                 |
| DNOP            | =        | 164000       | Display No Operation         |
| STATSA          | =        | 170000       | Load Status A<br>Instruction |
| LPLITE          | =        | 200          | Light Pen Hit On             |
| LPDARK          | =        | 300          | Light Pen Hit Off            |
| ITAL0           | =        | 40           | Italics Off                  |
| ITAL1           | =        | 60           | Italics On                   |
| SYNC            | =        | 4            | Halt and Resume in<br>Sync   |
| STATSB          | =        | 174000       | Load Status B<br>Instruction |



## Display File Handler

| <u>Mnemonic</u>            |   | <u>Value</u> | <u>Function</u>           |
|----------------------------|---|--------------|---------------------------|
| INCR                       | = | 100          | Graphplot Increment       |
| <u>(Vector/Point Mode)</u> |   |              |                           |
| INTX                       | = | 40000        | Intensity Vector or Point |
| MAXX                       | = | 1777         | Maximum X Component       |
| MAXY                       | = | 1377         | Maximum Y Component       |
| MINUSX                     | = | 20000        | Negative X Component      |
| MINUSY                     | = | 20000        | Negative Y Component      |
| <u>(Short Vector Mode)</u> |   |              |                           |
| SHIFTX                     | = | 200          |                           |
| MAXSX                      | = | 17600        | Maximum X Component       |
| MAXSY                      | = | 77           | Maximum Y Component       |
| MISVX                      | = | 20000        | Negative X Component      |
| MISVY                      | = | 100          | Negative Y Component      |

## N.8 ASSEMBLY INSTRUCTIONS

### N.8.1 General Instructions

All programs can be assembled in 16K, using RT-11 MACRO. All assemblies and all links should be error free. The following conventions are assumed:

1. Default extensions are not explicitly typed. These are .MAC for source files, .OBJ for assembler output, and .SAV for Linker output.
2. The default device (DK) is used for all files in the example command strings.
3. Listings and link maps are not generated in the example command strings.

### N.8.2 VTBASE

To assemble VTBASE with RT-11 link-up capability:

```
.R MACRO
*VTBASE=VTBASE
```



## Display File Handler

### N.8.3 VTCAL1 - VTCAL4

To assemble the modules VTCAL1 through VTCAL4:

```
.R MACRO
*VTCAL1=VTCAL1
*VTCAL2=VTCAL2
*VTCAL3=VTCAL3
*VTCAL4=VTCAL4
```

### N.8.4 VTHDLR

To create the concatenated handler module:

```
.R PIP
*VTHDLR.OBJ=VTCAL1.OBJ,VTCAL2.OBJ,VTCAL3.OBJ,VTCAL4.OBJ,VTBASE.OBJ/B
```

### N.8.5 Building VTLIB.OBJ

To build the VTLIB library:

```
.R LIBR
*VTLIB=VTHDLR.OBJ
```

## N.9 VTMAC

```
.NLIST
; VTMAC
; LIBRARY OF MACRO CALLS AND MNEMONIC DEFINITIONS
; FOR THE VT11 DEVICE SUPPORT PACKAGE

; DEC-11-OVTMA-8-LA

; COPYRIGHT (C) 1974
; DIGITAL EQUIPMENT CORPORATION
; MAYNARD, MASSACHUSETTS 01754

; MAY 1974

; THE INFORMATION IN THIS DOCUMENT IS SUBJECT TO
; CHANGE WITHOUT NOTICE AND SHOULD NOT BE CONSTRUED
; AS A COMMITMENT BY DIGITAL EQUIPMENT CORPORATION.
; DEC ASSUMES NO RESPONSIBILITY FOR ANY ERRORS THAT
; MAY APPEAR IN THIS DOCUMENT.

; DEC ASSUMES NO RESPONSIBILITY FOR THE USE
; OR RELIABILITY OF ITS SOFTWARE ON EQUIPMENT
; WHICH IS NOT SUPPLIED BY DEC.

; THIS SOFTWARE IS FURNISHED TO PURCHASER UNDER A
; LICENSE FOR USE ON A SINGLE COMPUTER SYSTEM AND
; CAN BE COPIED (WITH INCLUSION OF DEC'S COPYRIGHT
; NOTICE) ONLY FOR USE IN SUCH A SYSTEM, EXCEPT AS MAY
; OTHERWISE BE PROVIDED IN WRITING BY DEC.
```



## Display File Handler

```
; VTMAC IS A LIBRARY OF MACRO CALLS WHICH PROVIDE SUPPORT  
; OF THE VT11 DISPLAY PROCESSOR. THE MACROS PRODUCE CALLS  
; TO THE VT11 DEVICE SUPPORT PACKAGE, USING GLOBAL REFER-  
; ENCES.
```

```
; MACRO TO GENERATE A MACRO WITH ZERO ARGUMENTS.
```

```
.MACRO MAC0 NAME,CALL  
.MACRO NAME  
.GLOBL CALL  
JSR X'07,CALL  
.ENDM  
.ENDM
```

```
; MACRO TO GENERATE A MACRO WITH ONE ARGUMENT
```

```
.MACRO MAC1 NAME,CALL  
.MACRO NAME ARG  
.IF NB,ARG  
MOV ARG,X'00  
.ENDC  
.GLOBL CALL  
JSR X'07,CALL  
.ENDM  
.ENDM
```

```
; MACRO TO GENERATE A MACRO WITH TWO OPTIONAL ARGUMENTS
```

```
.MACRO MAC2 NAME,CALL  
.MACRO NAME ARG1,ARG2  
.GLOBL CALL  
.IF NB,ARG1  
MOV ARG1,X'00  
.ENDC  
.IF NB,ARG2  
MOV ARG2,-(X'06)  
.IFF  
CLR -(X'06)  
.NARG T  
.IF EQ,T  
CLR X'00  
.ENDC  
.ENDC  
JSR X'07,CALL  
.ENDM  
.ENDM
```



# Display File Handler

; MACRO LIBRARY FOR VT11:

|      |                   |
|------|-------------------|
| MAC0 | <,CLEAR>,<SVINIT> |
| MAC0 | <,STOP>,<SVSTOP>  |
| MAC0 | <,START>,<SVSTRT> |
| MAC0 | <,SYNC>,<SSYNC>   |
| MAC0 | <,NOSYN>,<SNOSYN> |
| MAC0 | <,UNLNK>,<SVUNLK> |
| MAC1 | <,INSRT>,<SVNSRT> |
| MAC1 | <,REMOV>,<SVRMOV> |
| MAC1 | <,BLANK>,<SVBLNK> |
| MAC1 | <,RESTR>,<SVRSTR> |
| MAC1 | <,STAT>,<SVSTPM>  |
| MAC1 | <,LPEN>,<SVLPEN>  |
| MAC1 | <,SCROL>,<SVSCRL> |
| MAC1 | <,NAME>,<SNAME>   |
| MAC2 | <,TRACK>,<SVTRAK> |
| MAC0 | <,LNKRT>,<SVRTLK> |

; MNEMONIC DEFINITIONS FOR THE VT11 DISPLAY PROCESSOR

|               |                                      |
|---------------|--------------------------------------|
| ;             |                                      |
| DJMP=160000   | ;DISPLAY JUMP                        |
| DNOP=164000   | ;DISPLAY NOP                         |
| DJSR=173400   | ;DISPLAY SUBROUTINE CALL             |
| DRET=173400   | ;DISPLAY SUBROUTINE RETURN           |
| DNAME=173520  | ;SET NAME REGISTER                   |
| DSTAT=173420  | ;RETURN STATUS DATA                  |
| DHALT=173500  | ;STOP DISPLAY AND RETURN STATUS DATA |
| ;             |                                      |
| CHAR=100000   | ;CHARACTER MODE                      |
| SHORTV=104000 | ;SHORT VECTOR MODE                   |
| LONGV=110000  | ;LONG VECTOR MODE                    |
| POINT=114000  | ;POINT MODE                          |
| GRAPHX=120000 | ;GRAPH X MODE                        |
| GRAPHY=124000 | ;GRAPH Y MODE                        |
| RELATV=130000 | ;RELATIVE VECTOR MODE                |
| ;             |                                      |
| INT0=2000     | ;INTENSITY 0                         |
| INT1=2200     |                                      |
| INT2=2400     |                                      |
| INT3=2600     |                                      |
| INT4=3000     |                                      |
| INT5=3200     |                                      |
| INT6=3400     |                                      |
| INT7=3600     |                                      |
| ;             |                                      |
| LPOFF=100     | ;LIGHT PEN OFF                       |
| LPON=140      | ;LIGHT PEN ON                        |
| BLKOFF=20     | ;BLINK OFF                           |
| BLKON=30      | ;BLINK ON                            |
| LINE0=4       | ;SOLID LINE                          |
| LINE1=5       | ;LONG DASH                           |
| LINE2=6       | ;SHORT DASH                          |
| LINE3=7       | ;DOT DASH                            |
| ;             |                                      |



# Display File Handler

```

STATSA=170000 ;LOAD STATUS REG A
LPLITE=200 ;INTENSIFY ON LPEN HIT
LPDARK=300 ;DON'T INTENSIFY
ITAL0=40 ;ITALICS OFF
ITAL1=60 ;ITALICS ON
SYNC=4 ;POWER LINE SYNC
;
STATSB=174000 ;LOAD STATUS REG B
INCR=100 ;GRAPH PLOT INCREMENT
INTX=40000 ;INTENSIFY VECTOR OR POINT
MAXX=1777 ;MAXIMUM X INCR. - LONGV
MAXY=1377 ;MAXIMUM Y INCR. - LONGV
MINUSX=20000 ;NEGATIVE X INCREMENT
MINUSY=20000 ;NEGATIVE Y INCREMENT
MAXSX=17600 ;MAXIMUM X INCR. - SHORTV
MAXSY=77 ;MAXIMUM Y INCR. - SHORTV
MISVX=20000 ;NEGATIVE X INCR. - SHORTV
MISVY=100 ;NEGATIVE Y INCR. - SHORTV
;LIST

```

## N.10 EXAMPLES USING GTON

EXAMPLE #1 RT-11 MACRO VM02-06 9-AUG-74 PAGE 4

```

2 ;.TITLE EXAMPLE #1
3 ;
4 ; THIS EXAMPLE USES THE .LPEN STATUS BUFFER AND THE
5 ; NAME REGISTER TO MODIFY A DISPLAY FILE WITH THE LIGHT PEN.
6 ;
7 000000 R0=X0
8 000001 R1=X1
9 000007 PC=X7
10 000044 JSW=44 ;JOB STATUS WORD
11
12 .MCALL .TTINR,.EXIT,.PRINT
13 00000 START: .LNKRT ;LINK TO MONITOR
14 00004 100004 BPL 15 ;LINK UP ERROR?
15 00006 .PRINT #EMSG ;YES, PRINT MESSAGE
16 00014 .EXIT ;AND EXIT.
17 00016 15: .SCROL #SCBUF ;ADJUST SCROLL
18 00026 .PRINT #MSG
19 00034 .INSRT #DFILE ;INSERT DISPLAY FILE
20 00044 .LPEN #LBUF ;SET UP LPEN BUFFER
21 00054 052737 BIS #100,#JSW ;SET JSW FOR TTINR
21 000100
21 000044
22 00062 005767 LTST: TST LBUF ;LIGHT PEN HIT?
22 000070
23 00066 001003 BNE 15 ;YES
24 00070 .TTINR ;NO, ANY TT INPUT?
25 00072 103025 BCC EXIT ;YES, EXIT
26 00074 000772 BR LTST ;NO, LOOP AGAIN
27 00076 016777 15: MOV I2,#IPTR ;RESTORE PREVIOUS CODE
27 000074
27 000102
28 00104 016701 MOV LBUF+2,R1 ;GET NAME VALUE

```



# Display File Handler

```

28      000050
29 00110 005301      DEC      R1      ;SUBTRACT ONE
30 00112 006301      ASL      R1      ;MULTIPLY BY TWO
31 00114 006701      ADD      PC,R1    ;USE TO INDEX
32 00116 0062701     ADD      #DTABL=.,R1 ;OFF TABLE DTABL.
32      000062
33 00122 011167      MOV      (R1),IPTR ;MOVE ADDR INTO IPTR
33      000060
34 00126 016777      MOV      I1,#IPTR ;MODIFY THAT CODE
34      000042
34      000052
35 00134 005067      CLR      LBUF     ;CLEAR BUFFER FLAG TO
35      000016
36
37 00140 000750      BR       LTST     ;ENABLE ANOTHER LP HIT.
38 00142 022700 EXIT:  CMP      #12,R0  ;LOOP AGAIN
38      000012      ;LINE FEED?
39 00146 001345      BNE      LTST     ;NO, GET ANOTHER
40 00150      .UNLNK
41 00154      .EXIT      ;UNLINK FROM MONITOR
42 00156      LBUF:      .BLKW      7      ;LPEN STATUS BUFFER
43 00174 103370 I1:   .WORD      CHAR!INT5!BLKON!LPON
44 00176 103160 I2:   .WORD      CHAR!INT4!BLKOFF!LPON
45 00200 000252!DTABL: .WORD      D1,D2,D3 ;TABLE OF DISPLAY FILE
45 00202 000272!

```

EXAMPLE #1 RT-11 MACRO VM02-06 9-AUG-74 PAGE 4+

```

45 00204 000312!
46
47 00206 000252!IPTR: .WORD      D1      ;LOCATIONS TO BE MODIFIED
48 00210 000002 SBUF: .WORD      2      ;PREVIOUS LOCATION MODIFIED
49 00212 001000      .WORD      1000    ;SCROLL LINE COUNT
50 00214      041 MSG: .ASCIZ    /!ERROR!/ ;SCROLL TOP Y POS.
50 00215      105      ;ERROR MESSAGE
50 00216      122
50 00217      122
50 00220      117
50 00221      122
50 00222      041
50 00223      000
51      .EVEN
52 00224      105 MSG: .ASCIZ    /EXAMPLE #1/ ;I.D. MESSAGE
52 00225      130
52 00226      101
52 00227      115
52 00230      120
52 00231      114
52 00232      105
52 00233      040
52 00234      043
52 00235      061
52 00236      000
53      .EVEN
54      ;
55      ; DISPLAY FILE FOR EXAMPLE #1
56      ;
57 00240 114000 OFILE: POINT
58 00242 000100      100
59 00244 000500      500
60 00246 173520      DNAME
61 00250 000001      1
62 00252 103160 D1:   CHAR!BLKOFF!INT4!LPON

```



# Display File Handler

```

63 00254      117      .ASCII /ONE./
63 00255      116
63 00256      105
63 00257      056
64 00260 114000      POINT
65 00262 000100      100
66 00264 000300      300
67 00266 173520      DNAME
68 00270 000002      2
69 00272 103160 D2:   CHAR|BLKOFF|INT4|LPON
70 00274      124      .ASCII /TWO./
70 00275      127
70 00276      117
70 00277      056
71 00300 114000      POINT
72 00302 000100      100
73 00304 000100      100
74 00306 173520      DNAME
75 00310 000003      3
76 00312 103160 D3:   CHAR|BLKOFF|INT4|LPON
77 00314      124      .ASCII /THREE./
77 00315      110

```

EXAMPLE #1 RT=11 MACRO VM02-06 9-AUG-74 PAGE 4+

```

77 00316      122
77 00317      105
77 00320      105
77 00321      056
78 00322 173400      DRET
79 00324 000000      0
80      000000      .END      START

```

EXAMPLE #1 RT=11 MACRO VM02-06 9-AUG-74 PAGE 4+

## SYMBOL TABLE

|                  |                   |                   |
|------------------|-------------------|-------------------|
| BLKOFF = 000020  | BLKON = 000030    | CHAR = 100000     |
| DFILE = 000240R  | DHALT = 173500    | DJMP = 160000     |
| DJSR = 173400    | DNAME = 173520    | DNOP = 164000     |
| DRET = 173400    | DSTAT = 173420    | DTABL = 000200R   |
| D1 = 000252R     | D2 = 000272R      | D3 = 000312R      |
| EMSG = 000214R   | EXIT = 000142R    | GRAPHX = 120000   |
| GRAPHY = 124000  | INCR = 000100     | INTX = 040000     |
| INT0 = 002000    | INT1 = 002200     | INT2 = 002400     |
| INT3 = 002600    | INT4 = 003000     | INT5 = 003200     |
| INT6 = 003400    | INT7 = 003600     | IPTR = 000206R    |
| ITAL0 = 000040   | ITAL1 = 000060    | I1 = 000174R      |
| I2 = 000176R     | JSW = 000044      | LBUF = 000156R    |
| LINE0 = 000004   | LINE1 = 000005    | LINE2 = 000006    |
| LINE3 = 000007   | LONGV = 110000    | LPDARK = 000300   |
| LPLITE = 000200  | LPOFF = 000100    | LPON = 000140     |
| LTST = 000062R   | MAXSX = 017600    | MAXSY = 000077    |
| MAXX = 001777    | MAXY = 001377     | MINUSX = 020000   |
| MINUSY = 020000  | MISVX = 020000    | MISVY = 000100    |
| MSG = 000224R    | PC = %000007      | POINT = 114000    |
| RELATV = 130000  | R0 = %000000      | R1 = %000001      |
| SCBUF = 000210R  | SHORTV = 104000   | START = 000000R   |
| STATSA = 170000  | STATSB = 174000   | SYNC = 000004     |
| SVLPEN = ***** G | \$VNSRT = ***** G | \$VRTLK = ***** G |
| SVSCHL = ***** G | \$VUNLK = ***** G |                   |

```

. ABS. 000000      000
      000326      001

```

```

ERRORS DETECTED: 0
FREE CORE: 15117. WORDS

```

MANEX1,LP:VTMAC,MANEX1



# Display File Handler

EXAMPLE #2 RT-11 MACRO VM02-06 9-AUG-74 PAGE 4

```

2          .TITLE  EXAMPLE #2
3
4          ; THIS EXAMPLE USES THE TRACKING OBJECT AND THE TRACK
5          ; COMPLETION ROUTINE TO CAUSE A VECTOR TO FOLLOW
6          ; THE LIGHT PEN FROM A SET POINT AT (500,500).
7
8          000000      R0=X0
9          000001      R1=X1
10         000006      SP=X6
11         000007      PC=X7
12
13         000000      .MCALL  .EXIT,.TTYIN,.PRINT
14         000004 100004  START: .LNKRT      ;LINK TO MONITOR
15         000006      BPL      1$          ;LINK UP ERROR?
16         000014      .PRINT  #EMSG       ;YES, INFORM USER
17         000016      .EXIT              ;AND EXIT
18         000026      1$: .INSRT  #DFILE    ;INSERT DISPLAY FILE
19         000042 004767 .TRACK  #TBUF,#TCOM ;DISPLAY TRACK OBJECT
19         000006      JSR      PC,WAIT     ;WAIT FOR <CR>
20
21         000046      .UNLNK              ;UNLINK FROM MONITOR
22         000052      .EXIT
23         000054      WAIT: .TTYIN         ;GET CHAR. FROM TTY
24         000060 022700 CMP      #12,R0    ;LINE FEED?
25         000064 001373      BNE      WAIT  ;NO, GET ANOTHER
26         000066 000207      RTS      PC
27         000070 000500 TBUF: .WORD      500,500 ;TRACK BUFFER INITED TO
28         000072 000500                                     ;START TRACK AT (500,500)
29
30         ; TRACK COMPLETION ROUTINE ENTERED AT INTERRUPT LEVEL
31         ; FROM DISPLAY FILE HANDLER WITH DISPLAY STOPPED.
32         ; USED TO UPDATE DISPLAY FILE WITH DATA FROM TBUF.
33
34         000074 010146 TCOM: MOV      R1,=(SP) ;SAVE R1
35         000076 016701      MOV      TBUF,R1 ;NEW X
36         000078 177766
37         00102 166701      SUB      0X,R1    ;NEW X - OLD X
38         000052
39         00106 100003      BPL      1$       ;POSITIVE DIFFERENCE?
40         00110 005401      NEG      R1       ;NO, SO MAKE POSITIVE
41         00112 052701      BIS      #MINUSX,R1 ;BUT SET MINUS BIT
42         00116 052701 1$: BIS      #INTX,R1  ;ALSO SET INTENSIFY BIT
43         00122 010167      MOV      R1,DX    ;THEN STORE IN DFILE.
44         000040
45         00126 016701      MOV      TBUF+2,R1 ;NEW Y
46         00132 166701      SUB      0Y,R1    ;NEW Y - OLD Y
47         000024
48         00136 100003      BPL      2$       ;POSITIVE DIFFERENCE?
49         00140 005401      NEG      R1       ;NO,SO MAKE POSITIVE
50         00142 052701      BIS      #MINUSX,R1 ;AND SET MINUS BIT
51         00146 010167 2$: MOV      R1,OY    ;THEN STORE IN DFILE
52         000016

```



# Display File Handler

EXAMPLE #2 RT-11 MACRO VM02-06 9-AUG-74 PAGE 4+

```

47 00152 012001      MOV      (SP)+,R1      ;RESTORE R1
48 00154 000207      RTS      PC           ;EXIT FROM COMPLETION ROUTINE
49
50                  / DISPLAY FILE FOR EXAMPLE #2
51                  /
52 00156 114000  DFIL:  POINT              ;SET POINT AT
53 00160 000500  OX:   500
54 00162 000500  OY:   500                ;(500,500)
55 00164 113000      LONGV,INT4          ;DRAW A VECTOR
56 00166 000000  DX:   .WORD 0           ;INITIALLY NOWHERE
57 00170 000000  DY:   .WORD 0
58 00172 173400      DRET                  ;DISPLAY FILE END
59 00174 000000      0
60 00176      123  MSG:  .ASCIZ  /SORRY, THERE SEEMS TO BE A PROBLEM/
60 00177      117
60 00200      122
60 00201      122
60 00202      131
60 00203      054
60 00204      040
60 00205      124
60 00206      110
60 00207      105
60 00210      122
60 00211      105
60 00212      040
60 00213      123
60 00214      105
60 00215      105
60 00216      115
60 00217      123
60 00220      040
60 00221      124
60 00222      117
60 00223      040
60 00224      102
60 00225      105
60 00226      040
60 00227      101
60 00230      040
60 00231      120
60 00232      122
60 00233      117
60 00234      102
60 00235      114
60 00236      105
60 00237      115
60 00240      000
61
62      0000001  .EVEN      .END      START

```



# Display File Handler

EXAMPLE #2  
SYMBOL TABLE

RT=11 MACRO VM02-06 9-AUG-74 PAGE 4+

|                         |                 |                 |
|-------------------------|-----------------|-----------------|
| BLKOFF# 000020          | BLKON # 000030  | CHAR # 100000   |
| DFILE 000156R           | DHALT # 173500  | DJMP # 160000   |
| DJSR # 173400           | DNAME # 173520  | DNOP # 164000   |
| DRET # 173400           | DSTAT # 173420  | DX 000166R      |
| DY 000170R              | EMSG 000176R    | GRAPHX# 120000  |
| GRAPHY# 124000          | INCR # 000100   | INTX # 040000   |
| INT0 # 002000           | INT1 # 002200   | INT2 # 002400   |
| INT3 # 002600           | INT4 # 003000   | INT5 # 003200   |
| INT6 # 003400           | INT7 # 003600   | ITAL0 # 000040  |
| ITAL1 # 000060          | LINE0 # 000004  | LINE1 # 000005  |
| LINE2 # 000006          | LINE3 # 000007  | LONGV # 110000  |
| LPDARK# 000300          | LPLITE# 000200  | LPOFF # 000100  |
| LPON # 000140           | MAXSX # 017600  | MAXSY # 000077  |
| MAXX # 001777           | MAXY # 001377   | MINUSX# 020000  |
| MINUSY# 020000          | MISVX # 020000  | MISVY # 000100  |
| OX 000160R              | OY 000162R      | PC #X000007     |
| POINT # 114000          | RELATV# 130000  | R0 #X000000     |
| R1 #X000001             | SHORTV# 104000  | SP #X000006     |
| START 000000R           | STATSA# 170000  | STATSB# 174000  |
| SYNC # 000004           | TBUF 000070R    | TCUM 000074R    |
| WAIT 000054R            | SVNSRT# ***** G | SVRTLK# ***** G |
| SVTRAK# ***** G         | SVUNLK# ***** G |                 |
| . ABS. 000000 000       |                 |                 |
| 000242 001              |                 |                 |
| ERRORS DETECTED: 0      |                 |                 |
| FREE CORE: 15022. WORDS |                 |                 |
| MANEX2,LP1=VTMAC,MANEX2 |                 |                 |







## APPENDIX O

### BATCH

This appendix is reserved for the RT-11  
BATCH documentation, which will be avail-  
able at a later date.







## APPENDIX P

### ERROR MESSAGE SUMMARY

This appendix summarizes in alphabetical order all error messages which may occur under the RT-11 System. The appropriate program for reference is included for the user's convenience. Error types are grouped into the following categories:

|         |  |
|---------|--|
| ACTION  | Error message is sent to the terminal and execution stops. A user-response is expected before execution can continue.  |
| COUNT:n | Error message is sent to the terminal; execution continues until nth occurrence of error, at which time the error is treated as FATAL. (For FORTRAN messages only.)  |
| FATAL   | Error message is sent to the terminal or the listing output file; the current command or statement is ignored, or execution is terminated, in which case the user must enter another command.  |
| IGNORE  | A subcategory of INFORMATIONal, which is used for FORTRAN messages only. Error is detected but no message is sent to the terminal or the listing output file, and execution continues.   |
| INFORM  | Informational message; an error condition is detected and the user is informed, either at the terminal or in the listing file; execution continues. The error condition may affect execution at a later time, and may require future action. |
| WARNING | A subcategory of INFORMATIONal; error message is sent to the terminal or the listing output file, and execution continues totally.   |



# Error Message Summary

| <u>MESSAGE</u>                              | <u>PROGRAM</u> | <u>ERROR<br/>TYPE</u> | <u>MEANING</u>   |
|---|----------------|-----------------------|--|
| A   | MACRO          | INFORM                | Addressing error. An address within the instruction was incorrect. Also may indicate a relocation error. The addition of two relocatable symbols is flagged as an A error. May also indicate that a local symbol was defined more than 128 words from the beginning of a local symbol block. |
| ADDITIVE REF OF xxxxxx<br>AT SEGMENT yyyyyy | LINKER         | WARNING               | Rule 1 of overlay rules explained in Section 6.6 has been violated. xxxxxx represents the entry point; yyyyyy represents the segment number.   |
| ?ADDR?                                      | MONITOR        | FATAL                 | Address out of range in E or D command.  |
| ?ADDR NOT IN SEG?                           | PATCH          | FATAL                 | The address was not in the specified segment.  |
| ADJUSTABLE DIMENSIONS ILLEGAL FOR ARRAY     | FORTTRAN       | WARNING               | ****<br>Adjustable arrays must be a dummy argument in a subprogram, and the adjustable dimensions must be integer dummy arguments in the subprogram. Violation of this rule causes a dimension of 1 to be used.  |
|   | FORTTRAN       | INFORM                | All arrays must be dimensioned with integer constants except as specified in the RT-11 FORTRAN COMPILER AND OBJECT TIME SYSTEM USER'S MANUAL (DEC-11-LRFPA-A-D).   |
| ?ARG ARGUMENT ERROR AT LINE xxxxx           | BASIC          | FATAL                 | Arguments in a function call do not match (in number or in type) the argument defined for the function.  |
| ARRAY **** HAS TOO MANY DIMENSIONS          | FORTTRAN       | INFORM                | An array can have up to seven dimensions.  |
| ?ATL ARRAYS TOO LARGE AT LINE xxxxx         | BASIC          | FATAL                 | There was not enough room in the memory available for the arrays specified in the DIM statements.  |



# Error Message Summary

|                                    |          |        |   |
|------------------------------------|----------|--------|---|
| ATTEMPT TO EXTEND COMMON BACKWARDS | FORTTRAN | INFORM | While attempting to equivalence arrays in COMMON, an attempt was made to extend COMMON past the recognized beginning of COMMON storage.   |
| B                                  | MACRO    | INFORM | Bounding error. Instructions or word data would be assembled at an odd address in memory. The location counter is updated by +1.  |
| ***** B                            | FORTTRAN | INFORM | Columns 1-5 of a continuation line were not blank. (Columns 1-5 of a continuation line must be blank except for a possible 'D' in column 1); the columns are ignored and compilation continues. |
| ?/B NO VALUE?                      | LINKER   | FATAL  | The /B switch requires an octal number as an argument.  |
| ?/B ODD VALUE?                     | LINKER   | FATAL  | The argument to the /B switch was not an unsigned even octal number.  |
| ?BAD CHECKSUM?                     | PATCHO   | FATAL  | A formatted binary block in the input file had a checksum which did not agree with that calculated for its contents.  |
| ?BAD GSD?                          | LINKER   | FATAL  | There was an error in the global symbol directory. (GSD). The file is probably not a legal object module. This error message occurs on pass 1 of the Linker.                                    |
| ?BAD LIBR?                         | LIBR     | FATAL  | The user attempted to build a library file containing no directory entries or gave an illegally constructed library file to the Librarian as input.   |
| BAD MACRO ARG                      | EXPAND   | INFORM | The macro argument was not formatted correctly.   |
| ?BAD OBJ?                          | LIBR     | FATAL  | A bad object module was detected during input.  |
|                                    | PATCHO   | FATAL  | The input file contained information which could be interpreted as an object module.  |



# Error Message Summary

|  |         |  |  |
|--|---------|--|--|
| BAD OVERLAY AT SEG #yyyyyy<br>LINKER       | WARNING | Overlay tried to store text outside of its region; check for an .ASECT in overlay. yyyyyy represents the segment number.   |  |
| ?BAD PATCH?                      PATCHO    | INFORM  | Checksum entered on exit does not agree with that calculated by PATCHO but the patch was made as specified. This probably indicates a typing error.                    |  |
| ?BAD PPN?                          FILEX   | FATAL   | The DOS/BATCH user identification code was not in the form [nnn,nnn], where each nnn is an octal number less than or equal to 377 (octal).                             |  |
| ?BAD RLD?                          LINKER  | FATAL   | There was an invalid relocation directory (RLD) command in the input file; the file is probably not a legal object module. The message occurs on pass 2 of the Linker. |  |
| ?BAD SWITCH?                      EXPAND   | FATAL   | An unrecognized command string switch was specified.   |  |
|  | LINKER  | FATAL  | LINK did not recognize a switch specified on the first command line. On a subsequent command line, a bad switch causes this warning message but does not restart the Linker. |
|  | MACRO   | FATAL  | The switch specified was not recognized by the program.  |
|  | PATCH   | FATAL  | Typed a switch other than /O or /M.  |
| ?BAD x SWITCH IGNORED?<br>LINKER           | WARNING | LINK did not recognize a switch (x) specified in the command line. The switch is ignored and linking continues.  |  |
| ?BDR BAD DATA READ AT LINE xxxxxx<br>BASIC | FATAL   | Item input from data segment list by READ statement was bad.   |  |
| BExxxxxx                          ODT      | FATAL   | Bad breakpoint entry from location xxxxxx. Breakpoint routine was entered for no known breakpoint; illegal trace trap instruction; T-bit was set in status register,   |  |



# Error Message Summary

|  |          |         |   |
|--|----------|---------|---|
|  |          |         | or a jump to the middle of ODT occurred.  |
| ?BOOT COPY?                                | PIP      | FATAL   | An error occurred during an attempt to write bootstrap with /U switch.  |
| ?BOTTOM ADDR WRONG? PATCH                  |          | FATAL   | The bottom address specified or contained in location 42 of an overlay file was incorrect. Specify the correct one using the b:B command.   |
| BRT BAD DATA-RETYPE FROM ERROR             | BASIC    | ACTION  | Item entered to input statement was bad.  |
| ?BSO BUFFER STORAGE OVERFLOW at line xxxxx | BASIC    | FATAL   | Not enough room available in file buffers.  |
| ?BSW?                                      | ASEMBL   | FATAL   | The switch specified was not recognized by the program.   |
| BYTE RELOCATION ERROR AT xxxxxx            | LINKER   | WARNING | Linker attempted to relocate and link byte quantities, but failed. xxxxxx represents the address at which the error occurred. Failure is defined as the high byte of the relocated value (or the linked value) not being all zero. In such a case, the value is truncated to 8 bits and the Linker continues processing (for save image and LDA files only; byte relocation is completely illegal for REL files). |
| C  | FORTTRAN | FATAL   | Constant subscript overflow. Too many subscripts have been employed in a statement.<br><br>SOLUTION - Simplify the statement.   |
| ***** C                                    | FORTTRAN | INFORM  | Illegal continuation. Comments cannot be continued and the first line of a program unit cannot be a continuation line. The continued comment is ignored and compilation continues.  |
| ?C-CHAIN-ONLY-CUSP? CREF                   |          | FATAL   | An attempt was made to either "R CREF" or to "START" a copy of CREF which was in memory due to a previous MACRO run   |



# Error Message Summary

|                                   |          |        |   |
|-----------------------------------|----------|--------|---|
|                                   |          |        | which had chained to CREF but was aborted from the console terminal (via CTRL C). CREF can only be "chained" to; it cannot be invoked without MACRO.                  |
| ?C-CRF FILE ERROR?                | CREF     | FATAL  | An output error occurred while accessing "DK:CREF.TMP", the temporary file passed from MACRO to CREF.   |
| ?C-DEVICE?                        | CREF     | FATAL  | An invalid device was specified to CREF by MACRO (system error).  |
| ?C-LST FILE ERROR?                | CREF     | FATAL  | An input/output error occurred while attempting to write the cross-reference table to the listing file.   |
| *CB ALMOST FULL*                  | EDIT     | ACTION | The command currently being entered is within 10 characters of exceeding the space available in the Command Buffer.   |
| ?CB FULL?                         | EDIT     | INFORM | Command exceeded the space allowed for a command string in the Command Buffer. The last 10 characters entered are ignored.  |
| ?CHK SUM?                         | PIP      | INFORM | A checksum error occurred in a formatted binary transfer.   |
| COMMON BLOCK EXCEEDS MAXIMUM SIZE | FORTTRAN | INFORM | An attempt was made to allocate more space to COMMON than is physically addressable (>32K words).   |
| ?COR OVR?                         | FILEX    | FATAL  | There was insufficient main storage for buffers and input list expansion. Try copying files one at a time, without using the "wild-card" (*.*) construction on input. |
|                                   | PIP      | FATAL  | Memory overflow--too many devices and/or file specifications (usually *. operations) and no room for buffers.   |
|                                   | SRCCOM   | FATAL  | Not enough memory to hold a particular difference section.  |



# Error Message Summary

|   |         |        |   |
|---|---------|--------|---|
| ?CORE?                                  | ASEMBL  | FATAL  | There were too many symbols in the program being assembled. Try dividing program into separately-assembled subprograms.   |
|   | LINKER  | FATAL  | There was not enough memory to accommodate the command or the resultant load module.  |
| ?CSECT ERROR?                           | LIBR    | FATAL  | The user has extended beyond the allowable .CSECT space for an object module to be placed in the library (i.e., the object module contains greater than 127(decimal) .CSECTs).            |
| CTn: PUSH REWIND OR MOUNT NEW VOLUME    | PIP     | ACTION | A new cassette must be mounted on drive n to allow continuation of an I/O operation. The operation is continued automatically as soon as the new cassette is mounted.                     |
| D                                       | MACRO   | INFORM | Doubly-defined symbol referenced. Reference was made to a symbol which was defined more than once.  |
| DANGLING OPERATOR                       | FORTRAN | INFORM | An operator (+, -, *, /, etc.) was missing an operand. Example: (I=J+).   |
| ?DAT?                                   | MONITOR | FATAL  | The DATE command argument was illegal, or no argument was given and the date has not yet been set.  |
| ?DCE DEVICE CHANNEL ERROR AT LINE xxxxx | BASIC   | FATAL  | The device channel number specified for a sequential or virtual memory file was out of range (1-7), or an attempt was made to open a virtual memory file on a non-file structured device. |
| DEFECTIVE DOTTED KEYWORD                | FORTRAN | INFORM | A dotted relational operator was not recognized. Also, possible misuse of decimal point.  |
| ?DEV FUL?                               | CSI     | FATAL  | Output file did not fit.  |
|   | PIP     | FATAL  | No room on device for file.   |



# Error Message Summary

|   |          |        |   |
|---|----------|--------|---|
| ?DEV FULL?                              | FILEX    | FATAL  | There was no room in the directory for the filename or there was no room on the output device for the file (in which case the filename is not placed in the output device directory). |
|   | LIBR     | FATAL  | The device was full; LIBR was unable to create or update the indicated library file. The CSI prints an asterisk and waits for the user to enter another command line.                 |
| ?DIR ERR?                               | FILEX    | FATAL  | An error occurred while reading or looking up the directory of the input device, or the input device does not have the proper file structure.   |
| ?*DIR FULL*?                            | EDIT     | FATAL  | No room in device directory for output file.  |
| ?DNR DEVICE NOT READY                   | BASIC    | FATAL  | An OLD command reads a file which did not have any BASIC statements.  |
| DO TERMINATOR *** PRECEDES DO STATEMENT | FORTTRAN | INFORM | The statement specified as the terminator of a DO loop must come after the DO statement.  |
| ?DUMP ERROR?                            | PATCHO   | FATAL  | An input/output error occurred while dumping a module.  |
| ?DV0 DIVISION BY 0 AT LINE xxxxx        | BASIC    | FATAL  | Program attempted to divide some quantity by 0.   |
| E                                       | MACRO    | INFORM | End directive not found. (A .END is generated.)   |
| ***** E                                 | FORTTRAN | INFORM | Missing END statement. An END statement is supplied by the Compiler if end-of-file is encountered.  |
| ?*** ERROR *** text                     | EXPAND   |        | Indicates nonfatal errors in the input file; "text" is a short explanation of the error; refer to individual text listing.  |



# Error Message Summary

|  |          |       |   |
|--|----------|-------|---|
| ?*EOF*?  | EDIT     | FATAL | Attempted a Read, Next or file searching command and no data was available.   |
| ↑ER ↑ERROR AT LINE xxxxx<br>BASIC                  |          | FATAL | The program tried to compute the value A↑B, where A<0 and B was not an integer. This produces a complex number which is not represented in BASIC. x>87 in EXP(x) function.  |
| ?ER RD DIR?  | PIP      | FATAL | Unrecoverable error reading directory. Check volume for off-line or write-locked condition and try operation again.   |
| ?ER RD OVLY?                                       | MONITOR  | FATAL | An I/O error occurred while reading a KMON overlay to process the current command. This is a serious error, indicating that the system file MONITR.SYS is unreadable.   |
| ?ER WR DIR?  | PIP      | FATAL | Unrecoverable error writing directory. Try again.   |
| ?"<"ERR?   | EDIT     | FATAL | Iteration brackets were nested too deeply or used illegally or brackets were not matched.   |
| ?ERR nn text                                       | FORTTRAN |       | Error diagnostics detected by the Object Time System are reported in either short form where nn is a decimal error condition, or long form where "text" is an added short error description. Refer to individual decimal error.                     |
| ?ERROR ERROR?                                      | LINKER   | FATAL | An error occurred while the Linker was in the process of recovering from a previous system or user error.   |
| ?ERROR IN FETCH?                                   | LINKER   | FATAL | The device is not available.  |
| ?ETC EXPRESSION TOO COMPLEX AT LINE xxxxx<br>BASIC |          | FATAL | The expression being evaluated caused the stack to overflow (usually because the parentheses were nested too deeply). The degree of complexity that produces this error varies according to the amount of space available in the stack at the time. |



## Error Message Summary

|                                       |          |        |   |
|---------------------------------------|----------|--------|---|
|                                       |          |        | Breaking the statement into several simpler ones eliminates the error.  |
| EXPECTING LEFT PARENTHESES AFTER **** | FORTTRAN | INFORM |   |
|                                       |          |        | An array name or function name reference was not followed by a left parenthesis.  |
| ?EXT NEG?                             | PIP      | FATAL  | A /T command attempted to make file smaller.  |
| EXTRA CHARACTER AT END OF STATEMENT   | FORTTRAN | INFORM |   |
|                                       |          |        | All the necessary information for a syntactically correct FORTRAN statement has been found on this line, but more information exists. Possibly due to inadvertent continuation signal on next line, or a missing comma. |
| F?                                    | MONITOR  | FATAL  | A CTRL F was typed and no foreground job exists.  |
| FATAL ERROR n                         | FORTTRAN |        | Diagnostics in this format report hardware error conditions and conditions which may require rewriting the program. n is a single letter representing an error code; refer to the individual listing of n.              |
| ?F ACTIVE?                            | MONITOR  | FATAL  | Neither FRUN nor UNLOAD may be used when a foreground job already exists and is active.   |
| ?FDE FILE DATA ERROR AT LINE xxxxxx   | BASIC    | FATAL  |   |
|                                       |          |        | Tried to write an element on an integer virtual memory file outside the range (x)<32,768.   |
| ?FEATURE NOT IMP?                     | FILEX    | FATAL  | An operation was attempted which FILEX cannot perform (e.g., zeroing an RT-11 device).  |
| ?FG ACTIVE?                           | FILEX    | FATAL  | An attempt was made to use /T when a foreground job was active. The transfer is not allowed until the foreground job is terminated and unloaded via UNLOAD FG.  |
| ?FG PRESENT?                          | PIP      | FATAL  | An attempt was made to use /O or /S while a foreground job  |



# Error Message Summary

|                                   |         |        |   |
|-----------------------------------|---------|--------|---|
|                                   |         |        | was still in memory. Unload it if it is no longer desired.  |
| ?FIL NAM?                         | FILEX   | FATAL  | The output filename was invalid or null.  |
| ?FIL NOT FND?                     | CSI     | FATAL  | Input file was not found.   |
|                                   | FILEX   | FATAL  | The input file was not found, or the "wild-card" (*,*) construction matched none of the existing files.   |
|                                   | LIBR    | FATAL  | One of the input files indicated in the command line was not found. The CSI prints an asterisk; the command may be reentered.   |
|                                   | MONITOR | FATAL  | File specified in R, RUN, GET, or FRUN command not found.   |
|                                   | PIP     | FATAL  | File not found during a delete, copy, or rename operation.  |
| ?FILE?                            | MONITOR | FATAL  | No file named where one was expected.   |
| ?FILNAM.EXT ALREADY EXISTS?       | FILEX   | FATAL  | An attempt was made to create the named file (FILNAM.EXT) on a DOS DECTape when a file already existed under the name specified. Use /D to delete the file, and retry the transfer. |
| ?*FILE FULL*?                     | EDIT    | FATAL  | Available space on output file was full. Type a CTRL C and a CLOSE monitor command to save the data already written.  |
| ?FILE NOT FND?                    | LINKER  | FATAL  | Input file was not found.   |
| ?*FILE NOT FND*?                  | EDIT    | FATAL  | Attempted to open a nonexistent file for editing.   |
| %FILES ARE DIFFERENT              | SRCCOM  | INFORM | Indicates that the files compared were different.   |
| ?FIO FILE I/O ERROR AT LINE xxxxx | BASIC   | FATAL  | An I/O error occurred. All files are automatically closed.  |



# Error Message Summary

|   |          |        |   |
|---|----------|--------|---|
| FLOATING CONSTANT TOO SMALL                 | FORTTRAN | FATAL  | A floating constant in an expression was too close to zero to be represented in the internal format. Use zero if possible.  |
| ?FNF FILE NOT FOUND AT LINE xxxxx           | BASIC    | FATAL  | The file requested was not found on the specified device.   |
| ?FNO FILE NOT OPEN AT LINE xxxxx            | BASIC    | FATAL  | The sequential or virtual memory file referenced was not open.  |
| ?FORLIB NOT FND?                            | LINKER   | FATAL  | The user indicated via the /F switch that the FORTRAN library, FORLIB, was to be linked with the other object modules in the command line, and the Linker could not find FORLIB.OBJ on the system device. |
| ?FTS FILE TOO SHORT AT LINE xxxxx           | BASIC    | FATAL  | The sequential file space allocated to an output file was inadequate.   |
| ?FWN FOR WITHOUT NEXT AT LINE xxxxx         | BASIC    | FATAL  | The program contained a FOR statement without a corresponding NEXT statement to terminate the loop.   |
| ?GND GOSUBS NESTED TOO DEEPLY AT LINE xxxxx | BASIC    | FATAL  | Program GOSUBs nested to more than 20 levels.   |
| ***** H                                     | FORTTRAN | INFORM | Hollerith string or quoted literal string longer than 255 characters or longer than the remainder of the statement. The statement is aborted.   |
| ?HANDLR?                                    | MONITOR  | FATAL  | Attempted to close a file with no handler in memory. The file cannot be closed.   |
| ?HARD I/O ERROR?                            | LINKER   | FATAL  | A hardware error has occurred; try again.   |
| ?*HDW ERR*?                                 | EDIT     | FATAL  | A hardware error occurred during I/O. May be caused by WRITE LOCKED device. Try again.  |
| I   | MACRO    | INFORM | Illegal character detected. Illegal characters which are also non-printing are  |



# Error Message Summary

|             |                           |        |  |
|-------------|---------------------------|--------|--|
| ***** I     |                           |        | replaced by a ? on the listing. The character is then ignored.   |
|             | FORTTRAN                  | INFORM | Non-FORTTRAN character used. The line contains a character that is not in the FORTRAN character set and is not in a Hollerith string or comment line. The character is ignored and compilation continues.  |
| ?IDF        | ILLEGAL DEF AT LINE xxxxx |        |  |
|             | BASIC                     | FATAL  | The DEF statement contained an error.  |
| ?IDM        | ILLEGAL DIM AT LINE xxxxx |        |  |
|             | BASIC                     | FATAL  | Syntax error in a dimension statement.   |
| ?ILL ARG?   |                           |        |  |
|             | EDIT                      | FATAL  | The argument specified was illegal for the command used. A negative argument was specified where a positive one was expected or argument exceeds the range + or - 16,383.  |
| ?ILL ASECT? |                           |        |  |
|             | LINKER                    | FATAL  | The user has attempted to place an .ASECT above 1000 in a foreground link or to place an .ASECT into an overlay foreground link.   |
| ?ILL CMD?   |                           |        |  |
|             | CSI                       | FATAL  | Syntax error.  |
|             | EDIT                      | FATAL  | EDIT did not recognize the command line specified; ED was not the first command issued when used to activate the display hardware.   |
|             |                           |        |  |
|             | FILEX                     | FATAL  | The length of the command line exceeded 72 characters; the command line was not in proper CSI format; the UIC exceeded the allowed number of characters or [ was used without ]; a "wild-card" (*,*) construction was used on a non-file structured device; no output or no input file was specified for a copy operation; or more than one filename construction (dev:filnam.ext) was specified on either side of the = (<) sign. |



# Error Message Summary

|                  |         |       |   |
|------------------|---------|-------|---|
|                  | LIBR    | FATAL | An illegal command was used in the command line. The CSI prints an asterisk; the command may be reentered.  |
|                  | MONITOR | FATAL | Illegal Keyboard Monitor command or command line too long.  |
|                  | PIP     | FATAL | The command specified was not syntactically correct; a device name is missing which should be specified, a switch argument is too large, a filename is specified where one is inappropriate, or a non-file structured device is specified for a file-structured operation.  |
| xxxxxx ?ILL DEL? | LIBR    | FATAL | An attempt was made to delete from the library's directory a module or an entry point that does not exist; xxxxxx represents the module or entry point name. The CSI prints an asterisk and waits for the user to enter another command line.   |
| ?ILL DEV?        | CSI     | FATAL | Device specified does not exist.  |
|                  | FILEX   | FATAL | The device handler was not found, an invalid device name was used, or one of the following was attempted:<br><br>RK or DT was not used for DOS/BATCH (RSTS-11) input in a copy operation;<br><br>DT was not used for DOS/BATCH (RSTS-11) output in a zero or delete operation;<br><br>DT was not used for DOS/BATCH (RSTS-11) output in a copy operation;<br><br>DT was not used for DECsystem-10 input in any operation. |
|                  | LIBR    | FATAL | An illegal device was specified in the command line. The CSI prints an asterisk; the command may be reentered.  |



# Error Message Summary

|                           |                |                |  |
|---------------------------|----------------|----------------|--|
|                           | MONITOR<br>PIP | FATAL<br>FATAL | Illegal or nonexistent device, or an attempt was made to make a device handler resident for use with a foreground job (dev=F) when the Single-Job Monitor was running.   |
| ?*ILL DEV*?               | EDIT           | FATAL          | Attempted to open a file on an illegal device, or attempted to use display hardware when none was available (it may be in use by the other job).   |
| xxxxxx ?ILL INS?          | LIBR           | FATAL          | An attempt was made to insert a module into a library which contains the same entry point as an existing module. xxxxxx represents the entry point name. The CSI prints an asterisk; a new command may be entered. |
| ?ILL MAC?                 | EDIT           | FATAL          | Delimiters were improperly used, or an attempt was made to enter an M command during execution of a Macro or an EM command while an EM was in progress.  |
| ?*ILL NAME*?              | EDIT           | FATAL          | Filename specified in EB, EW, or ER was illegal.   |
| ILL REN?                  | PIP            | FATAL          | Illegal rename operation. Usually caused by different device names on the input and output sides of the command string.  |
| xxxxxx ?ILL REPL?         | LIBR           | FATAL          | An attempt was made to replace in the library file a module which does not already exist. xxxxxx represents the module name. The CSI prints an asterisk and waits for the user to enter another command line.      |
| ?ILL SWT?                 | FILEX          | FATAL          | An illegal switch was used in a command line (e.g., a switch not listed in Table J-1).   |
|                           | PIP            | FATAL          | Illegal switch or switch combination.  |
| ILLEGAL ADJACENT OPERATOR | FORTTRAN       | INFORM         | Two operators (*, /, logical operators, etc.) were   |



## Error Message Summary

|  |   |
|--|---|
| <p>?ILLEGAL COMMAND?      PATCHO      FATAL</p>                          | <p>illegally placed next to each other. Example: I/*J.</p> <p>A command line was not recognized, or it was not in the proper format for the particular command.</p> |
| <p>ILLEGAL DO STATEMENT TERMINATOR ****<br/>FORTRAN      INFORM</p>      | <p>A DO statement terminator must not be a GO TO, arithmetic IF, RETURN, or DO statement or logical IF containing one of these statements.</p>                      |
| <p>ILLEGAL ELEMENT IN I/O LIST<br/>FORTRAN      INFORM</p>               | <p>An item, expression, or implied DO specifier in an I/O list was of illegal syntax.</p>   |
| <p>ILLEGAL STATEMENT ON LOGICAL IF<br/>FORTRAN      INFORM</p>           | <p>The statement contained in a logical IF must not be another logical IF or DO statement.</p>  |
| <p>ILLEGAL TYPE FOR OPERATOR<br/>FORTRAN      INFORM</p>                 | <p>An illegal variable type has been used with an exponentiation or logical operator.</p>   |
| <p>ILLEGAL USAGE OR MISSING LEFT PARENTHESIS<br/>FORTRAN      INFORM</p> | <p>A left parenthesis was required but not found, or a variable reference or constant was illegally followed by a left parenthesis.</p>                             |
| <p>?ILN ILLEGAL NOW      BASIC      FATAL</p>                            | <p>An attempt was made to execute an INPUT statement in immediate mode.</p>   |
| <p>?ILR ILLEGAL READ AT LINE xxxxx<br/>BASIC      FATAL</p>              | <p>Tried to open a write-only device for input or tried to read on a sequential file open for output.</p>   |
| <p>?IN ER?      PIP      FATAL</p>                                       | <p>Unrecoverable error reading file. Try again (this error is ignored during /C operation).</p>   |
| <p>?IN ERR?      DUMP      FATAL</p>                                     | <p>A hardware error occurred while reading the input file and /G was not specified in the command line.</p>   |



# Error Message Summary

|  |        |       |   |
|--|--------|-------|---|
|  | FILEX  | FATAL | A device error occurred on input.   |
|  | LIBR   | FATAL | An unrecoverable hardware/software error has occurred while processing an input file. The CSI prints an asterisk and waits for another command to be entered.   |
|  | SRCCOM | FATAL | A hardware error occurred in reading input.   |
| IN LINE nnnnn MSG# m text<br>FORTRAN     |        |       | Errors reported by the secondary phase of the compiler. nnnnn is the internal sequence number of the statement, m specifies the error number, text is explanation of error; see individual listings for text.   |
| IN LINE %WARNING% MSG# m text<br>FORTRAN |        |       | Indicates a condition which may be potentially dangerous at execution time or which may present compatibility problems with other FORTRAN compilers. m specifies the error number and text is explanation of error; see individual listings for text. |
| ?INCORRECT FILE SPEC?                    | PATCH  | FATAL | The response to the "FILE NAME--" message was not of the correct form. Try again.   |
| ?INPUT ERROR?                            | EXPAND | FATAL | Hardware error in reading an input file.  |
| ?INSUFFICIENT CORE?                      | EXPAND | FATAL | Not enough memory to store macro definitions.   |
|  | MACRO  | FATAL | There were too many symbols in the program being assembled. Try dividing program into separately-assembled sub-programs.  |
|  | PATCH  | FATAL | PATCH did not have enough memory to hold the file's device handler plus the internal "segment table". This message should not occur.  |



## Error Message Summary

|  |         |        |   |
|--|---------|--------|---|
| INTEGER OVERFLOW                             | FORTRAN | INFORM | An integer constant or expression value must not fall outside the range -32767 to +32767.                                   |
| INVALID COMPLEX CONSTANT                     | FORTRAN | INFORM | A complex constant has been improperly formed.  |
| INVALID DIMENSIONS FOR ARRAY ****            | FORTRAN | INFORM | An attempt was made while dimensioning an array to explicitly specify zero as one of the dimensions.                        |
| INVALID DO TERMINATOR ORDERING AT LABEL **** | FORTRAN | INFORM | DO loops were incorrectly nested.   |
| INVALID EQUIVALENCE                          | FORTRAN | INFORM | Illegal equivalence, or equivalence that was contradictory to a previous equivalence.                                       |
| INVALID FORMAT SPECIFIER                     | FORTRAN | INFORM | A format specifier was not the label of a FORMAT statement or an array name.  |
| INVALID IMPLICIT RANGE SPECIFIER             | FORTRAN | INFORM | Illegal implicit range specifier; i.e., non-alphabetic specifier, or specifier range was in reverse alphabetic order.       |
| INVALID LOGICAL UNIT                         | FORTRAN | INFORM | A logical unit reference must be an integer variable or constant in the range 1 to 99.                                      |
| INVALID OCTAL CONSTANT                       | FORTRAN | INFORM | An octal constant was too large or contained a digit other than 0-7.  |
| INVALID OPTIONAL LENGTH SPECIFIER            | FORTRAN | INFORM | A data type declaration optional length specifier was illegal. For example, REAL*4 and REAL*8 are legal, but REAL*6 is not. |
| INVALID RADIX50 CONSTANT                     | FORTRAN | INFORM | Illegal character detected in a RADIX50 constant.   |
| INVALID RECORD FORMAT                        | FORTRAN | INFORM | The third parenthetical argument in a DEFINE FILE statement must be a single character U.                                   |
| ?INVALID RELOC REG? PATCH                    |         | FATAL  | Tried to reference a relocation register outside the range 0-7.   |



# Error Message Summary

|  |         |        |   |
|--|---------|--------|---|
| ?INVALID SEG NO?                             | PATCH   | FATAL  | The segment number S: did not exist.  |
| INVALID STATEMENT IN BLOCK DATA              | FORTRAN | INFORM | It is illegal to have any executable or FORMAT statements in a BLOCK DATA subprogram.   |
| INVALID STATEMENT LABEL REFERENCE            | FORTRAN | INFORM | Reference has been made to a statement number that is of illegal construction. GO TO 999999 is illegal since the statement number is too long.            |
| INVALID SUBROUTINE OR FUNCTION NAME          | FORTRAN | INFORM | A name used in a CALL statement or function reference was not valid. Example: use of an array name in a CALL statement subroutine name reference.         |
| INVALID TARGET FOR ASSIGNMENT                | FORTRAN | INFORM | The left side of an arithmetic assignment statement was not a variable name or array element reference.   |
| INVALID TYPE SPECIFIER                       | FORTRAN | INFORM | An unrecognizable data type was used.   |
| INVALID USAGE OF FUNCTION OR SUBROUTINE NAME | FORTRAN | INFORM | A function name cannot appear in a DIMENSION, COMMON, DATA, EQUIVALENCE, or Data Type Declaration statement.  |
| INVALID VARIABLE NAME                        | FORTRAN | INFORM | A variable name was missing, contained an illegal character, or did not begin with an alphabetic character.   |
| ?I/O ERROR ON CHANNEL n?                     | MACRO   | FATAL  | A hardware error occurred while attempting to read from or write to the device on the channel specified in the message.                                   |
| ***** K                                      | FORTRAN | INFORM | Illegal statement label definition. Illegal (non-numeric) character in statement label. The illegal statement label is ignored and compilation continues. |



# Error Message Summary

|                                |         |        |   |
|--------------------------------|---------|--------|---|
| L                              | MACRO   | INFORM | Line buffer overflow; i.e., input line greater than 132 characters. Extra characters on a line (more than 72 decimal) are ignored in terminal mode.   |
| ***** L                        | FORTRAN | INFORM | Line too long. There are more than 80 characters in a line; this diagnostic is issued before the line containing too many characters. The line is truncated to 80 characters and compilation continues. |
| LABEL ON DECLARATIVE STATEMENT |         |        |   |
|                                | FORTRAN | INFORM | It is illegal to place a label on a declarative statement.  |
| ?LDA FILE ERROR?               | LINKER  | FATAL  | There was a hardware problem with the device specified for LDA output or the device is full.  |
| ?LIBR FIL ILL REPL?            | LIBR    | FATAL  | The user has specified that a library file be replaced by another library file. Only object modules can be replaced.  |
| LINE TOO LONG                  | EXPAND  | INFORM | A line has become longer than 132 characters.   |
| ?LP NOT FND?                   | DUMP    | FATAL  | A line printer handler was not available on the system.   |
| ?LTL LINE TOO LONG             | BASIC   | FATAL  | The line typed was longer than 120 characters; the line buffer overflows.   |
| M                              | MACRO   | INFORM | Multiple definition of a label. A label was encountered which was equivalent (in the first six characters) to a previously encountered label.   |
| ?/M ODD VAL?                   | LINKER  | FATAL  | An odd value has been specified for the stack address. Control returns to the Linker and another command line may be indicated.   |
| ***** M                        | FORTRAN | INFORM | Multiply defined label. The label is ignored.   |



## Error Message Summary

|                       |         |        |  |
|-----------------------|---------|--------|--|
| MACRO ALREADY DEFINED | EXPAND  | INFORM | A macro was defined more than once.  |
| MACRO(S) NOT FOUND    | EXPAND  | INFORM | Macros listed in an .MCALL statement were not found in SYSMAC.8K (make sure SYSMAC.8K is present on system).   |
| ?M-BAD FETCH xx       | MONITOR | FATAL  | Either an error occurred while reading a device handler from SY, or the address at which the handler was to be loaded is illegal. xx represents the address plus 2 of the location where the error occurred.   |
| ?M-DIR IO ERR xx      | MONITOR | FATAL  | An error occurred doing I/O in the directory of a device (e.g., .ENTER on a write-locked device). xx represents the address plus 2 of the location where the error occurred.   |
| ?M-DIR OVLFO xx       | MONITOR | FATAL  | No more directory segments were available for expansion (occurs during creation via file .ENTER). xx represents the address plus 2 of the location where the error occurred.   |
| ?M-DIR UNSAFE xx      | MONITOR | FATAL  | In F/B only, this message may appear in addition to any of the other M- diagnostics listed. It indicates that the error occurred while the USR was updating a device directory. One or more files on that device may be lost. xx represents the address plus 2 of the location where the error occurred. |
| ?M-FP TRAP xx         | MONITOR | FATAL  | A floating-point exception trap occurred, and the user program had no .SFPA exception routine active (see Chapter 9). xx represents the address plus 2 of the location where the error occurred.   |
| ?M-ILL ADDR xx        | MONITOR | FATAL  | Under the F/B Monitor, an address specified in a monitor call was odd or was not within the job's limits. xx represents the address plus 2 of the location where the error occurred.   |



# Error Message Summary

|                                     |         |       |   |
|-------------------------------------|---------|-------|---|
| ?M-ILL CHAN xx                      | MONITOR | FATAL | A channel number was specified which was too large. xx represents the address plus 2 of the location where the error occurred.  |
| ?M-ILL EMT xx                       | MONITOR | FATAL | An EMT was executed which did not exist; i.e., the function code was out of bounds. xx represents the address plus 2 of the location where the error occurred.  |
| ?M-ILL USR xx                       | MONITOR | FATAL | The USR was called from a completion routine. This error does not have a soft return (i.e., .SERR will not inhibit this message; see Chapter 9). xx represents the address plus 2 of the location where the error occurred.   |
| ?M-NO DEV xx                        | MONITOR | FATAL | A READ/WRITE operation was tried but no device handler was in memory for it. xx represents the address plus 2 of the location where the error occurred.   |
| ?M-OVLY ERR xx                      | MONITOR | FATAL | A user program with overlays has failed to successfully read an overlay. xx represents the address plus 2 of the location where the error occurred.   |
| ?M-SWAP ERR xx                      | MONITOR | FATAL | A hard I/O error occurred while the system was attempting to swap KMON or the USR. This is usually caused by a write-locked system device. Under the S/J Monitor, this may cause the system to halt. xx represents the address plus 2 of the location where the error occurred. |
| ?M-TRAP TO 4 xx<br>?M-TRAP TO 10 xx | MONITOR | FATAL | The job has referenced illegal memory, an illegal instruction was used, etc. xx indicates where the failure occurred.   |
| ?MAP FILE ERROR?                    | LINKER  | FATAL | There was a hardware problem with the device specified for map output or the device was full.   |



## Error Message Summary

|  |        |   |
|--|--------|---|
| MISSING ASSIGNMENT OPERATOR<br>FORTRAN     | INFORM | The first operator seen in an arithmetic assignment statement was not an equal sign (=). Example: I+J=K.  |
| MISSING COMMA<br>FORTRAN                   | INFORM | The comma delimiter was expected but was not found. See the section of the FORTRAN Reference Manual that describes the general form of the statement in question.         |
| MISSING COMMA IN MACRO ARG<br>EXPAND       | INFORM | Found spaces or tabs within a macro argument; try using brackets around the argument; e.g., <arg with spaces>.  |
| MISSING DELIMITER IN EXPRESSION<br>FORTRAN | INFORM | Two operands have been placed next to each other in an expression, with no operator between them.   |
| MISSING DOT<br>EXPAND                      | INFORM | A macro name or argument name did not begin with a dot.   |
| ?MISSING END IN MACRO?<br>EXPAND           | FATAL  | End of input was encountered while storing a macro definition; probably missing an .ENDM.   |
| MISSING LABEL<br>FORTRAN                   | INFORM | A statement label was expected but not found. Example: ASSIGN J TO I. A valid statement label reference should precede 'TO' but does not.                                 |
| MISSING RIGHT PARENTHESIS<br>FORTRAN       | INFORM | A right parenthesis was expected but not found. Example: READ(5,100,). The first non-blank character after the format reference should be a right parenthesis but is not. |
| MISSING QUOTATION MARK<br>FORTRAN          | INFORM | In a FIND statement, the logical unit number and record number must be separated by a single quotation mark.  |
| MISSING VARIABLE<br>FORTRAN                | INFORM | A variable was expected but not found. Example: ASSIGN 100 TO 1. A variable name should follow the 'TO' but does not.   |



## Error Message Summary

|   |          |        |  |  |
|---|----------|--------|--|--|
| MISSING VARIABLE OR CONSTANT                |          |        |  |  |
|   | FORTTRAN | INFORM |  | Looked for an operand (variable or constant) but found a delimiter (comma, parenthesis, etc.). Example: WRITE(). A unit number should follow the open parenthesis, but a delimiter (close parenthesis) is encountered instead. |
| MODES OF VARIABLE **** AND DATA ITEM DIFFER |          |        |  |  |
|   | FORTTRAN | INFORM |  | The data type of each variable and its associated data list item must agree in a DATA statement.   |
| ?MODULE NOT FOUND?                          | PATCHO   | INFORM |  | The module requested in a POINT command was not found in the input file between the position of the file at the time of the point and the end.   |
| ?MORE THAN 15 CHANGES?                      |          |        |  |  |
|   | PATCHO   | FATAL  |  | Too many changes have been specified for a particular module. The patch should be broken up into several steps.  |
| ?MORE THAN 5 CSECTS REQUIRE CHANGE?         |          |        |  |  |
|   | PATCHO   | FATAL  |  | An attempt has been made to patch locations in too many different CSECTS. The patch should be made in several steps.   |
| MULT DEF OF xxxxxx                          | LINKER   | FATAL  |  | The symbol, xxxxxx, was defined more than once.  |
| MULTIPLE DECLARATION FOR VARIABLE ****      |          |        |  |  |
|   | FORTTRAN | INFORM |  | A variable cannot appear in more than one data type declaration statement or dimensioning statement.   |
| ?MUST OPEN WORD?                            | PATCH    | FATAL  |  | The @ command was typed when a byte location was open.   |
| ?MUST SPECIFY SEG?                          | PATCH    | FATAL  |  | The address referenced was not in the root section; a segment number S: must be used.  |
| N   | MACRO    | INFORM |  | Number containing 8 or 9 had decimal point missing.  |
| NAME DOESN'T MATCH                          |          |        |  |  |
|   | EXPAND   | INFORM |  | Optional name given in .ENDM directive did not match name given in corresponding .MACRO directive.   |



# Error Message Summary

|                 |                                  |                |  |
|-----------------|----------------------------------|----------------|--|
| ?NBF            | NEXT BEFORE FOR AT LINE<br>BASIC | xxxxx<br>FATAL | The NEXT statement corresponding to a FOR statement preceded the FOR statement.  |
| ?NER            | NOT ENOUGH ROOM AT LINE<br>BASIC | xxxxx<br>FATAL | There was not enough room on the selected device for the specified number of output blocks.  |
| NESTED MACROS   | EXPAND                           | INFORM         | A macro was defined or invoked within another macro.   |
| ?NIF            | ASEMBL                           | FATAL          | No input file was specified and there must be at least one input file.   |
| ?NO ADDR OPEN?  | PATCH                            | FATAL          | The <line feed>, ↑ or @ command was typed when no location was open.   |
| ?NO CLOCK?      | MONITOR                          | FATAL          | No KWill clock was available for the TIME command.   |
| ?NO CORE?       | LIBR                             | FATAL          | Available free memory has been used up, The current command is aborted and the CSI prints an asterisk; a new command may be entered. |
| ?NO FG?         | MONITOR                          | FATAL          | A SUSPEND, RSUME, or UNLOAD FG command was given, but no foreground job was in memory.   |
| ?*NO FILE*?     | EDIT                             | FATAL          | Attempted to read or write when no file was open.  |
| ?NO FILE OPEN   | PATCHO                           | FATAL          | An attempt was made to use a command other than DEC or HELP before an OPEN command was issued.                                       |
| ?NO INPUT?      | LINKER                           | FATAL          | No input files were specified.   |
| ?NO INPUT FILE? | EXPAND                           | FATAL          | There must be at least one input file.   |
|                 | MACRO                            | FATAL          | No input file was specified and there must be at least one input file.   |
| NO NAME         | EXPAND                           | INFORM         | A macro definition had no name.  |
| ?*NO ROOM*?     | EDIT                             | FATAL          | Attempted to Insert, Save, Unsave, Read, Next, Change or Exchange when there was not enough room in the                              |



# Error Message Summary

|   |         |         |  |
|---|---------|---------|--|
|   |         |         | appropriate buffer. Delete unwanted buffers to create more room or write text to the output file.  |
| ?*NO TEXT*?                             | EDIT    | FATAL   | Attempted to call in text from the Save Buffer when there was no text available.   |
| ?NO SYS ACTION?                         | PIP     | WARNING | The /Y switch was not included with a command specified on a .SYS file. The command is executed for all but the .SYS files. A *.* transfer is most likely to cause this message. |
| ?NO UFD?                                | FILEX   | FATAL   | The specified UFD was not found on the DOS input disk.   |
| NON-STANDARD STATEMENT ORDERING         | FORTRAN | WARNING | Non-adherence to statement ordering requirements may cause error conditions on other FORTRAN compilers.  |
| ?NOT IN PROGR BOUNDS?                   | PATCH   | FATAL   | Tried to open a location beyond the end of the file.   |
| ?NPR NO PROGRAM                         | BASIC   | FATAL   | The RUN command has been specified, but no program has been typed in.  |
| ?NSM NUMBERS AND STRINGS MIXED AT LINE  | BASIC   | FATAL   | xxxxx String and numeric variables may not appear in the same expression, nor may they be set equal to each other (as in A\$=2).   |
| NUMBER IN FORMAT STATEMENT NOT IN RANGE | FORTRAN | INFORM  | An integer constant in a FORMAT statement was greater than 255 or was zero.  |
| O                                       | FORTRAN | FATAL   | Unrecoverable error occurred while the Compiler was writing the object file (.OBJ). Possibly insufficient output file space.   |
|   | MACRO   | INFORM  | SOLUTION - Rectify hardware problem, or make more space for output.  |
| /O IGNORED                              | LINKER  | WARNING | Opcode error. Directive out of context.  |
|   |         |         | Overlays have been specified in the wrong order (see   |



## Error Message Summary

|                                |        |       |   |
|--------------------------------|--------|-------|---|
|                                |        |       | overlay restrictions); the overlay switch is ignored.   |
| ?ODD ADDRESS?                  | PATCH  | FATAL | Tried to open a word address which was odd. (Use "\".)  |
| ?ODD BOTTOM ADDR?              | PATCH  | FATAL | The bottom address specified or contained in location 42 of an overlay file was odd.  |
| ?ODF                           | ASEMBL | FATAL | No room to continue writing output; try to compress device with PIP.  |
| ?OFFSET?                       | PATCHO | FATAL | The offset supplied in a WORD or BYTE command was not an octal number, or was in improper format.   |
| ?OOD OUT OF DATA AT LINE xxxxx | BASIC  | FATAL | The data list was exhausted and a READ requested additional data.   |
| ?OUT ER?                       | PIP    | FATAL | Unrecoverable error writing file. Perhaps a hardware or checksum error; try recopying file. Also may be caused by an attempt to compress a larger device to a smaller one or by not enough room when creating a file. The system takes the largest space available and divides it in half before attempting to insert the file. Try the [] construction or /X switch. |
| ?OUT ERR?                      | DUMP   | FATAL | A hardware error occurred while writing an output file, or the output device was full.  |
|                                | FILEX  | FATAL | A device error occurred on output.  |
|                                | LIBR   | FATAL | An unrecoverable hardware/software error has occurred while processing an output file. The CSI prints an asterisk and waits for the user to enter another command.  |
|                                | SRCCOM | FATAL | A hardware error occurred in writing output file, or output device full.  |
| ?OUT FIL?                      | PIP    | FATAL | Illegal output file specification or missing output file.   |



# Error Message Summary

## ?OUTPUT DEVICE FULL?

|        |       |  |
|--------|-------|--|
| EXPAND | FATAL | No room to continue writing output. Try to compress the device with PIP. |
|--------|-------|--|

|       |       |  |
|-------|-------|--|
| MACRO | FATAL | No room to continue writing output. Try to compress device with PIP. |
|-------|-------|--|

|                |        |       |   |
|----------------|--------|-------|---|
| ?OUTPUT ERROR? | PATCHO | FATAL | A hardware error (or possibly a write-lock condition) occurred while attempting to write the output file. |
|----------------|--------|-------|---|

|                         |        |       |  |
|-------------------------|--------|-------|--|
| ?OUTPUT FILE TOO SMALL? | PATCHO | FATAL | The space allocated to the output file was too small. This may be corrected by compressing the device with PIP (if enough total space is free on the device), or by using another device for output. |
|-------------------------|--------|-------|--|

|               |        |       |                             |
|---------------|--------|-------|-----------------------------|
| ?OUTPUT FULL? | LINKER | FATAL | The output device was full. |
|---------------|--------|-------|-----------------------------|

|                             |       |       |   |
|-----------------------------|-------|-------|---|
| ?OVF OVERFLOW AT LINE xxxxx | BASIC | FATAL | The result of a computation was too large for the computer to handle. |
|-----------------------------|-------|-------|---|

|           |         |       |  |
|-----------|---------|-------|--|
| ?OVR COR? | MONITOR | FATAL | Attempted to GET or RUN a file that was too big. |
|-----------|---------|-------|--|

|   |          |       |  |
|---|----------|-------|--|
| P | FORTTRAN | FATAL | Optimizer push down overflow. Statement too complex, or too many common subexpressions occurred in one basic block of a program. |
|---|----------|-------|--|

SOLUTION - Simplify complex statements.

|       |        |   |
|-------|--------|---|
| MACRO | INFORM | Phase error. A label's definition of value varied from one pass to another. |
|-------|--------|---|

|         |          |        |   |
|---------|----------|--------|---|
| ***** P | FORTTRAN | INFORM | Statement contained unbalanced parentheses. The statement is aborted. |
|---------|----------|--------|---|

|  |          |        |  |
|--|----------|--------|--|
| P-SCALE FACTOR NOT IN RANGE -127 to +127 | FORTTRAN | INFORM | P-scale factors must fall in the range -127 to +127. |
|--|----------|--------|--|

|          |         |       |  |
|----------|---------|-------|--|
| ?PARAMS? | MONITOR | FATAL | Bad parameters were typed to the SAVE command. |
|----------|---------|-------|--|

|                               |          |        |  |
|-------------------------------|----------|--------|--|
| PARENTHESES NESTED TOO DEEPLY | FORTTRAN | INFORM | Group repeats in a FORMAT statement have been nested too deeply. |
|-------------------------------|----------|--------|--|



# Error Message Summary

|  |          |        |  |
|--|----------|--------|--|
| ?PROG HAS NO SEGS?                     | PATCH    | FATAL  | The file specified as an overlay file was not.   |
| ?PTB PROGRAM TOO BIG                   | BASIC    | FATAL  | The line just entered caused the program to exceed the user code area.   |
| ?PWF POWER FAIL AT LINE xxxxx          | BASIC    | FATAL  | A power fail occurred while the specified line was being executed.   |
| Q                                      | MACRO    | INFORM | Questionable syntax. There were missing arguments or the instruction scan was not completed or a carriage return was not immediately followed by a line feed or form feed.   |
| R                                      | FORTTRAN | FATAL  | Unrecoverable hardware error occurred while the Compiler was reading source file.  |
|  | MACRO    | INFORM | SOLUTION - Rectify hardware problem.   |
|  | MACRO    | INFORM | Register-type error. An invalid use of or reference to a register has been made.   |
| ?RBG RETURN BEFORE GOSUB AT LINE xxxxx | BASIC    | FATAL  | A RETURN was encountered before execution of a GOSUB statement.  |
| READ ERROR?                            | PATCH    | FATAL  | File I/O error in reading.   |
| ?REBOOT?                               | PIP      | INFORM | .SYS files have been transferred, renamed, compressed or deleted from the system device. It may be necessary to reboot the system. The actual reboot operation must not be performed until PIP returns with the prompting asterisk for the next command. |

REFERENCE TO INCORRECT TYPE OF LABEL \*\*\*\*  
 FORTTRAN INFORM

A statement label reference that should be a label on a FORMAT statement was not such a label, or a statement label reference that should be a label on an executable statement was not such a label.



# Error Message Summary

|  |          |         |   |
|--|----------|---------|---|
| REFERENCE TO UNDEFINED STATEMENT LABEL     |          |         |   |
|  | FORTTRAN | INFORM  | A reference has been made to a statement number that has not been defined anywhere in the program unit.                             |
| ?REL FILE ERR?                             | LINKER   | FATAL   | The Linker encountered a problem writing the REL file; try again.   |
| ?REL FIL I/O ER?                           | MONITOR  | FATAL   | Either the program requested was not a REL file or a hardware error was encountered trying to read or write the file.               |
| ?ROOM?                                     | PIP      | FATAL   | Insufficient space following file specified with a /T switch.   |
| S  | FORTTRAN | FATAL   | Subexpression stack overflow; statement too complex.  |
|  |          |         | SOLUTION - Simplify complex statements.   |
| ***** S                                    | FORTTRAN | INFORM  | Syntax error (multiple equal signs, etc.). Statement not of the general FORTRAN statement form. The statement is aborted.           |
| ?SAV FILE ERR?                             | LINKER   | FATAL   | The Linker encountered a problem writing the save image file; try again.  |
| ?SOB SUBSCRIPT OUT OF BOUNDS AT LINE xxxxx | BASIC    | FATAL   | The subscript computed was greater than 32,767 or was outside the bounds defined in the DIM statement.                              |
| ?*SRCH FAIL*?                              | EDIT     | FATAL   | The text string specified in a Get, Find or Position command was not found in the available data.                                   |
| ?SSO STRING STORAGE OVERFLOW AT LINE xxxxx | BASIC    | FATAL   | There was not enough memory available to store all the strings used in the program.   |
| ?STACK ADDRESS UNDEFINED OR IN OVERLAY?    | LINKER   | WARNING | The stack address specified by the /M switch was either undefined or in an overlay. The stack address is set to the system default. |
| STATEMENT NUMBER MUST BE UNLABELED         |          |         |   |
|  | FORTTRAN | INFORM  | A data, subroutine, function, block data, arithmetic statement function   |



## Error Message Summary

|  |          |        |  |
|--|----------|--------|--|
|  |          |        | definition, or declarative statement must not be labeled.  |
| STATEMENT TOO COMPLEX                          | FORTTRAN | INFORM | An arithmetic statement function had more than 10 dummy arguments, or the statement was too long to compile. Break it up into 2 or more smaller statements.  |
| ?STL STRING TOO LONG AT LINE xxxxx             | BASIC    | FATAL  | The maximum length of a string in a BASIC statement is 255 characters.   |
| SUBROUTINE OR FUNCTION STATEMENT MUST BE FIRST | FORTTRAN | INFORM | A SUBROUTINE, FUNCTION or BLOCK DATA statement, if present, must be the first statement in a program unit.   |
| SUBSCRIPT OF ARRAY **** NOT IN RANGE           | FORTTRAN | INFORM | Array subscripts that were constants or constant expressions were checked to see if they were within the bounds of the array's dimensions.   |
| ?SV FIL I/O ER?                                | MONITOR  | FATAL  | I/O error on .SAV file in SAVE (output) or R, RUN, GET, or FRUN (input) command.   |
| ?SWITCH ERROR?                                 | SRCCOM   | FATAL  | An invalid switch was found or a switch other than /L was given a value.   |
| ?SWT ERR?                                      | FILEX    | FATAL  | An attempt was made to use more than one /S or /T switch in a command line (only one is allowed); an attempt was made to use more than one transfer mode switch (/I, /P, /A) or more than one operation switch (/D, /L, /F, /Z) in a command line (only one of each is allowed). |
| ?SY I/O ER?                                    | MONITOR  | FATAL  | I/O error on system device (usually reading or writing scratch area).  |
| ?SYMBOL TABLE OVERFLOW?                        | LINKER   | FATAL  | There were too many global symbols used in the program. Retry the link using the /S switch. If the error still occurs, the link cannot take place.   |



## Error Message Summary

?SYN SYNTAX ERROR AT LINE xxxxx  
BASIC FATAL

The program has encountered an unrecognizable statement. Common examples of syntax errors are misspelled commands and unmatched parentheses, and other typographical errors.

Wrong number of arguments was used in a function, or illegal delimiter in a function.

SYNTAX EXPAND INFORM

A macro directive was not constructed correctly.

SYNTAX ERROR FORTRAN INFORM

Check the general form of the statement with the general form outlined in the FORTRAN LANGUAGE REFERENCE MANUAL section that describes the type of statement.

SYNTAX ERROR IN INTEGER OR FLOATING CONSTANT  
FORTRAN INFORM

An integer or floating constant has been incorrectly formed. For example, 1.23.4 is an illegal floating constant because it contains two decimal points.

T FORTRAN FATAL

Memory Overflow.

SOLUTION - Break up program into subprograms or compile on larger machine.

MACRO INFORM

Truncation error. A number generated more than 16 bits of significance or an expression generated more than 8 bits of significance during the use of the .BYTE directive.

TARGET MUST BE ARRAY  
FORTRAN INFORM

The third argument in an ENCODE or DECODE statement must be an array name.

?TIM? MONITOR FATAL

The TIME command argument is illegal.

?TLT LINE TOO LONG TO TRANSLATE  
BASIC FATAL

Lines are translated as entered, and the line just entered exceeded the area available for translation.



# Error Message Summary

|  |         |         |   |
|--|---------|---------|---|
| ?TMO?                                    | ASEMBL  | FATAL   | Too many output files were specified.   |
| TOO MANY ARGS                            | EXPAND  | INFORM  | A macro directive had more than 30 arguments.   |
| ?TOO MANY OUTPUT FILES?                  | MACRO   | FATAL   | Too many output files were specified.   |
|  | LINKER  | FATAL   | The Linker allows specification of only two output files.   |
| ?TOO MUCH DIFFERENCE?                    | SRCCOM  | FATAL   | More than 310 (octal) lines of difference between two files were found.   |
| TRANSFER ADDRESS UNDEFINED OR IN OVERLAY | LINKER  | WARNING | The transfer address was not defined or was in an overlay.  |
| U  | MACRO   | INFORM  | Undefined symbol. An undefined symbol was encountered during the evaluation of an expression. Relative to the expression, the undefined symbol is assigned a value of zero. |
| ***** U                                  | FORTRAN | INFORM  | Statement could not be identified as any legal FORTRAN statement. The statement is aborted.   |
| ?UFN UNDEFINED FUNCTION AT LINE xxxxx    | BASIC   | FATAL   | The function called was not defined by the program or was not loaded with BASIC.  |
| ?ULN UNDEFINED LINE NUMBER AT LINE xxxxx | BASIC   | FATAL   | The line number specified in an IF, GO TO or GOSUB statement did not exist anywhere in the program.   |
| UNDEF GLBLS                              | LINKER  | WARNING | A load map has been requested and there were undefined globals.   |
| UNDEFINED GLOBALS<br>xxxxxxx<br>xxxxxxx  | LINKER  | WARNING | The globals listed (xxxxxxx) were undefined.  |
| UNLABELED FORMAT STATEMENT               | FORTRAN | INFORM  | All FORMAT statements must be labeled.  |
| USAGE OF VARIABLE **** INVALID           | FORTRAN | INFORM  | An attempt was made to EXTERNAL a common variable,  |



## Error Message Summary

an array variable, or a dummy argument or an attempt was made to place in COMMON a dummy argument or external name.

VARIABLE \*\*\*\* INVALID IN ADJUSTABLE DIMENSION

FORTRAN INFORM

A variable used as an adjustment dimension must be an integer dummy argument in the subprogram unit.

VARIABLE \*\*\*\* IS NOT WORD ALIGNED

FORTRAN WARNING

Placing a non-LOGICAL\*1 variable or array after a LOGICAL\*1 variable or array in COMMON or equivalencing non-LOGICAL\*1 variables or arrays to logical\*1 variables or arrays may cause this condition. An attempt to reference the variable at runtime will cause an error condition.

VARIABLE \*\*\*\* NAME EXCEEDS SIX CHARACTERS

FORTRAN WARNING

A variable name of more than six characters was specified. The first six characters were used as the true variable name. Other FORTRAN Compilers may treat this as an error condition.

W

FORTRAN FATAL

Unrecoverable error occurred while the Compiler was writing listing file. Possibly, listing file space was not large enough.

SOLUTION - Rectify hardware problem, or make more space for listing file.

?WLO WRITE LOCKOUT AT LINE xxxxx

BASIC FATAL

Tried to open a read-only device for output, or tried to write on a sequential or virtual file opened for input only.

?WRITE ERROR?

PATCH FATAL

File I/O error in writing.

?WRONG NUMBER OF OUTPUT FILES?

EXPAND FATAL

There must be exactly one output file.

WRONG NUMBER OF SUBSCRIPTS FOR ARRAY \*\*\*\*

FORTRAN INFORM

An array reference did not have the same number of



# Error Message Summary

|                               |         |         |  |
|-------------------------------|---------|---------|--|
| Y                             | FORTRAN | FATAL   | subscripts as specified when the array was dimensioned.  |
|                               |         |         | Code generation stack overflow - statement too complex.  |
| Z                             | FORTRAN | FATAL   | SOLUTION - Simplify complex statements.  |
|                               |         |         | Compiler error.  |
|                               | MACRO   | INFORM  | SOLUTION - Report this error to your local software support representative. Please include program listing.  |
|                               |         |         | Instruction which was not compatible among all members of the PDP-11 family (11/05, 11/20, 11/40, 11/45).  |
| dev:/Z ARE YOU SURE?          | PIP     | ACTION  |  |
|                               |         |         | Asks for confirmation from the user before zeroing a device.   |
| 0 NON-FORTRAN ERROR CALL      | FORTRAN | FATAL   |  |
|                               |         |         | A TRAP has occurred with an unrecognizable error code.   |
| 1 INTEGER OVERFLOW            | FORTRAN | FATAL   |  |
|                               |         |         | During an arithmetic operation, an integer's value has exceeded 32767 in magnitude.  |
| 2 INTEGER ZERO DIVIDE         | FORTRAN | FATAL   |  |
|                               |         |         | During an integer mode arithmetic operation an attempt was made to divide by zero.   |
| 3 COMPILER GENERATED ERROR    | FORTRAN | FATAL   |  |
|                               |         |         | An attempt was made to execute a FORTRAN statement in which the compiler detected errors.  |
| 4 COMPUTED GO TO OUT OF RANGE | FORTRAN | WARNING |  |
|                               |         |         | The integer variable or expression in a computed GO TO statement was less than 1 or greater than the number of statement label references in the list. Control is passed to the next executable statement. |



# Error Message Summary

|                                       |          |         |   |
|---------------------------------------|----------|---------|---|
| 5 INPUT CONVERSION ERROR              | FORTTRAN | COUNT:3 | During a formatted input operation an illegal character was detected in an input field. A value of 0 is returned.   |
| 6 OUTPUT CONVERSION ERROR             | FORTTRAN | IGNORE  | During a formatted output operation the value of a particular number could not be output in the specified field length without loss of significant digits. The field is filled with '*'s. |
| 10 FLOATING OVERFLOW                  | FORTTRAN | COUNT:3 | During an arithmetic operation a real value has exceeded the largest representable real number. A value of 0 is returned.   |
| 11 FLOATING UNDERFLOW                 | FORTTRAN | IGNORE  | During an arithmetic operation a real value has become less than the smallest representable real number, and has been replaced with a value of zero.                                      |
| 12 FLOATING ZERO DIVIDE               | FORTTRAN | FATAL   | During a real mode arithmetic operation an attempt was made to divide by zero.  |
| 13 SQRT OF NEGATIVE NUMBER            | FORTTRAN | COUNT:3 | An attempt was made to take the square root of a negative number. The result is replaced by 0.  |
| 14 UNDEFINED EXPONENTIATION OPERATION | FORTTRAN | FATAL   | An attempt was made to perform an illegal exponentiation operation. For example $-3.**.5$ is illegal because the result would be an imaginary number.                                     |
| 15 LOG OF NEGATIVE NUMBER             | FORTTRAN | FATAL   | An attempt was made to take the logarithm of a negative number.   |
| 16 WRONG NUMBER OF ARGUMENTS          | FORTTRAN | FATAL   | One of the FORTRAN Library functions, or System Subroutines which checks for such an occurrence, has been called with an improper number of arguments.                                    |



## Error Message Summary

- |    |  |          |       |  |
|----|--|----------|-------|--|
| 20 | INVALID CHANNEL NUMBER                 | FORTTRAN | FATAL | An illegal logical unit number has been specified in an I/O statement. A logical unit number must be an integer between 1 and 99.  |
| 21 | NO AVAILABLE CHANNELS                  | FORTTRAN | FATAL | An attempt was made to have too many devices simultaneously open for I/O. The maximum number of active channels is six by default, but this number may be changed at compile time.   |
| 22 | INPUT RECORD TOO LONG                  | FORTTRAN | FATAL | During an input operation a record was encountered that was longer than the maximum record length. The default maximum record length is 133 (decimal) bytes, but this number may be altered at compile time.               |
| 23 | HARDWARE I/O ERROR                     | FORTTRAN | FATAL | A hardware error has been detected during an I/O operation.  |
| 24 | ATTEMPT TO READ/WRITE PAST END OF FILE | FORTTRAN | FATAL | If this occurred during a WRITE, more space was needed to contain the output file.   |
| 25 | ATTEMPT TO READ AFTER WRITE            | FORTTRAN | FATAL | An attempt was made to read after writing on a file. A WRITE must be followed by a REWIND or BACKSPACE before a read operation can be performed.   |
| 26 | RECURSIVE I/O NOT ALLOWED              | FORTTRAN | FATAL | An expression in the I/O list of a WRITE statement has caused initiation of another READ or WRITE operation. This can happen if a FUNCTION that performs I/O is referenced in an expression in a WRITE statement I/O list. |
| 27 | ATTEMPT TO USE DEVICE NOT IN SYSTEM    | FORTTRAN | FATAL | Self-explanatory.  |
| 28 | OPEN FAILED FOR FILE                   | FORTTRAN | FATAL | A file could not be found.   |



## Error Message Summary

|  |          |       |   |
|--|----------|-------|---|
| 29 NO ROOM FOR DEVICE HANDLER                    | FORTTRAN | FATAL | There was not enough free memory left to accommodate a specific device handler.                                     |
| 30 NO ROOM FOR BUFFERS                           | FORTTRAN | FATAL | There was not enough free memory left to set up required I/O buffers.   |
| 31 NO AVAILABLE RT-11 CHANNEL                    | FORTTRAN | FATAL | More than the maximum number of RT-11 channels, 15, were requested to be simultaneously opened for I/O.             |
| 32 FMTD-UNFMTD-RANDOM I/O TO SAME FILE           | FORTTRAN | FATAL | An attempt was made to perform any combination of formatted, unformatted, or random access I/O to the same file.    |
| 33 ATTEMPT TO READ PAST END OF RECORD            | FORTTRAN | FATAL | An attempt was made to read a larger record than actually existed in a file.  |
| 34 UNFMTD I/O TO TTY OR LPT                      | FORTTRAN | FATAL | An attempt was made to perform an unformatted write operation on the terminal or line printer.                      |
| 35 ATTEMPT TO OUTPUT TO READ ONLY FILE           | FORTTRAN | FATAL | Self-explanatory.   |
| 36 BAD FILE SPECIFICATION STRING                 | FORTTRAN | FATAL | Self-explanatory.   |
| 37 RANDOM ACCESS READ/WRITE BEFORE DEFINE FILE   | FORTTRAN | FATAL | A random access READ or WRITE operation was attempted before a DEFINE FILE was performed.                           |
| 38 RANDOM I/O NOT ALLOWED ON TTY OR LPT          | FORTTRAN | FATAL | Random access I/O was illegally attempted on the terminal or line printer.  |
| 39 RECORD LARGER THAN RECORD SIZE IN DEFINE FILE | FORTTRAN | FATAL | A record was encountered that was larger than that specified in the DEFINE FILE statement for a random access file. |



## Error Message Summary

- 40 REQUEST FOR A BLOCK LARGER THAN 65535  
FORTRAN FATAL An attempt was made to reference an absolute disk block address greater than 65535.
- 41 DEFINE FILE ATTEMPTED ON AN OPEN UNIT  
FORTRAN FATAL An open file must be closed before another DEFINE FILE can be performed on that unit.
- 42 MEMORY OVERFLOW COMPILING OBJECT TIME FORMAT  
FORTRAN FATAL The OTS has run out of free memory while scanning an array format that was generated at run time.
- 43 SYNTAX ERROR IN OBJECT TIME FORMAT  
FORTRAN FATAL A syntax error was encountered while the OTS was scanning an array format that was generated at run time.
- 44 2ND RECORD REQUEST IN ENCODE/DECODE  
FORTRAN FATAL ENCODE and DECODE will operate only on a single record at a time.
- 45 INCOMPATIBLE VARIABLE AND FORMAT TYPES  
FORTRAN FATAL An attempt was made to output a real variable with an integer field descriptor or an integer variable with a real field descriptor.
- 46 INFINITE FORMAT LOOP  
FORTRAN FATAL The format associated with an I/O statement that includes an I/O list has no field descriptors to use in transferring those variables.
- 47 ATTEMPT TO STORE OUTSIDE PARTITION  
FORTRAN FATAL In an attempt to store data into a subscripted variable, the address calculated for the array element in question did not lie within the section of memory allocated to the job. The subscript in question is out-of-bounds. (Message is issued only when bounds-checking modules have been installed in FORLIB.)
- 60 STACK OVERFLOWED  
FORTRAN FATAL The hardware stack has overflowed. Proper traceback may be impaired.



# Error Message Summary

## 61 ILLEGAL MEMORY REFERENCE

FORTRAN FATAL

This may be any type of BUS error, most probably an illegal memory address reference.

## 62 FORTRAN START FAIL

FORTRAN FATAL

The program has been loaded into memory but there was not enough free memory remaining for the OTS to initialize work space and buffers.

## 63 ILLEGAL INSTRUCTION

FORTRAN FATAL

The program has attempted to execute an illegal instruction, e.g., floating point arithmetic instruction on a machine with no floating point hardware.



## GLOSSARY

|                               |  |
|-------------------------------|--|
| Absolute address              | A binary number that is assigned as the address of a fixed memory storage location.  |
| Absolute block numbers        | Any blocks which use block 0 of a physical device as a base. Relative blocks use the start of a file as a base.  |
| Absolute section              | That portion of a program in which the programmer specifies the physical (absolute) locations of data items, using the .ASECT assembler directive.   |
| Addressing mode               | The portion of a PDP-11 machine instruction which specifies how the argument is to be referenced.  |
| Alphanumeric character string | A command string containing any of the 26 alphabetic characters A-Z, the numeric characters 0-9, space and certain special characters.   |
| Argument                      | A variable or constant which is given in the call of a subroutine as information to it; a variable upon whose value the value of a function depends; the known reference factor necessary to find an item in a table or array (i.e., the index). |
| Argument block                | A block of memory used to transmit programmed request arguments to the monitor.  |
| Array                         | A set or list of elements, usually subscripted variables or data.  |
| ASCII                         | American Standard Code for Information Interchange. Established by American National Standards Institute to standardize a binary code for alphanumeric characters.   |
| Assembler                     | A program that translated symbolic opcodes into machine language and assigns memory locations for variables and constants.   |
| Assembler directives          | The mnemonics used in the assembly language programs to control or direct the assembly process.  |
| Assembly language             | A symbolic language that translates directly into machine language instructions. Usually there is a one-to-one relation between assembly language instructions and machine language instructions.  |



|                    |   |
|--------------------|---|
| Asynchronous       | <ol style="list-style-type: none"> <li>1. Pertaining to the scheduling of hardware operations by ready and done signals rather than by time intervals.</li> <li>2. Pertaining to the method of data transmission in which each character is sent with its own synchronizing information.</li> </ol> |
| Autoincrement mode | Mode of operation wherein the contents of the register are incremented immediately after being used as address of the operand.  |
| Background program | A program operating automatically, under low priority, when higher priority (foreground) programs are not using system resources.   |
| Backup file        | A copy of a file created for protection in case the primary file is inadvertently destroyed.  |
| Bad blocks         | A defective block of storage on any type of magnetic storage medium that produces a hardware error when attempting to read or write in the block.   |
| Base address       | A given address from which an absolute address is derived by combination with a relative address. An address constant.  |
| Base segment       | The portion of an overlaid program that is always memory resident; same as root segment.  |
| Baud               | A unit of signaling speed. In a code in which all characters have the same length, one baud corresponds to a rate of one signal element per second, usually one bit per second.   |
| Binary             | Pertaining to the number system having a base or radix of two. In this system numbers are represented by 1s and 0s. Counting to ten in binary: 1, 10, 11, 100, 101, 110, 111, 1000, 1001, 1010.   |
| Bit                | Contraction of "Binary digit", a bit is the smallest unit of information in a binary system of notation. It is the choice between two possible states, usually designated one (1) and zero (0).   |
| Blocks             | A set of consecutive machine words, characters or digits handled as a unit, particularly in reference to I/O. Each type of mass storage has its own block size, its own smallest unit of storage (e.g. PDP-11 DECTape has 256 <sub>10</sub> 16-bit words per block). See Data Block.                |



|                         |   |
|-------------------------|---|
| Bootstrap               | A program of several instructions whose purpose is to load and (usually) start a complex system of programs.  |
| Bottom address          | The lowest memory address in which a program is located.  |
| Bounds                  | The limits a program may operate within.  |
| Breakpoint              | A preset point in a user program where control passes to a debugging routine.   |
| Buffer                  | A temporary storage area which may be a special register or an area of storage. Buffers are often used to hold data being passed between processes or devices which operate at different speeds or different times. |
| Byte                    | A group of binary digits usually operated upon as a unit (typically 1/2 word).  |
| Carry bit               | A bit in the program status word indicating a carry from the most significant bit in the operation; also a common method of indicating a program request failure.   |
| Central processing unit | The part of the computer containing the Arithmetic and Logical Unit, the Instruction Control Unit, timing generators and I/O interfaces of the basic system.  |
| Chaining                | A programming technique which involves dividing a routine into sections with each section terminated by a call to the next section.   |
| Channel number          | Logical identifier in range 0 to 255 <sub>10</sub> assigned to a file used by RT-11 Monitor. Data blocks in an open file may be referred to by channel number.  |
| Channel status word     | A word associated with each I/O channel in RT-11 that preserves the status of that channel.   |
| Character-oriented      | Referring to editing operations on a single character. See also Line-oriented.  |
| Checksum                | A value representing the sum of all bytes in a program. When the program is loaded, the sum of the bytes can be compared with the checksum to make sure that the entire program has been loaded correctly.          |
| Clock                   | A time-keeping or frequency-measuring device within the computer system.  |
| Closed location         | A location whose contents are not available for examination and change.   |



|                              |  |
|------------------------------|--|
| Co-resident overlay routines | Overlay routines that are simultaneously resident in memory.   |
| Command string               | A series of characters which specify the input/output devices, files, and switches.  |
| Command String Interpreter   | The portion of the RT-11 system software which accepts an ASCII string input and interprets the string as input and output files and switches.   |
| Completion routines          | An optional user-supplied routine that is called at the completion of an operation.  |
| Compress                     | To move into one area all the free (unused) blocks that are interspersed in the directory and files on a specified device.   |
| Concatenate                  | To combine many files into one file.   |
| Condition codes              | The four bits of the Program Status Word in the PDP-11 Processor that preserve the results (negative, zero, carry, overflow) of the instruction just completed.  |
| Configuration                | A particular selection of computer, peripherals, and interfacing equipment that are functioning together. A list of the devices and computers of a computer system.  |
| Console device               | The interface, or communication device, between the operator and the computer, which contains indicator lights, switches, knobs and sometimes a keyboard to permit manual operation of the device.   |
| Constant register            | A logical register in ODT or PATCH which is used to store an often-used constant.  |
| Context switching            | The saving of key registers and other memory areas prior to switching between jobs, as in timesharing or multiprogramming.   |
| Contiguous                   | Code which resides in memory or on a peripheral device immediately adjacent to other sections of code or data.   |
| Control Section (CSECT)      | A named, contiguous part of a program. CSECT's are denoted by the directives ".CSECT" and ".ASECT" in the MACRO Assembler Language. CSECT's are collected and assigned addresses by the Linker.  |
| Core memory                  | The main high-speed storage of a computer in which binary data is represented by the switching polarity of magnetic cores. Semi-conductor memory is higher speed than core memory, but semi-conductor memory is more expensive and volatile. |



|                              |   |
|------------------------------|---|
| Cross reference table (CREF) | A list of all or a subset of symbols in source program and statements where they were defined or used.  |
| Cursor                       | On a display device, a symbol that appears to indicate the location of the pointer.   |
| Data block                   | A logical grouping of data, usually associated with input or output. Typical data blocks involved in RT-11 are 256 words long.  |
| Data format                  | The form or structure of information, particularly in an I/O file. RT-11 has several standard data formats such as ASCII and formatted binary.  |
| Debug                        | To detect, locate, and correct mistakes in a program.   |
| Default                      | A specification assigned by system program when user specification is omitted.  |
| Delimiter                    | A character that separates, terminates and organizes elements of a statement or program.  |
| Device block                 | A section of code that specifies a physical device and filename for an RT-11 programmed request.  |
| Device directory listing     | A list of all files on a specified device. List contains all files with associated creation dates, total free blocks on device, and number of blocks used by files. Magtape and cassette directories omit some information. See Directory.            |
| Device handlers              | Routines that perform I/O for specific storage devices and translate logical block numbers to physical disk, tape or drum addresses. These routines also handle error recovery and provide device independence in conjunction with operating systems. |
| Direct assignment            | User definition of symbol and associated value.   |
| Directive                    | see Assembler directives  |
| Directory                    | An area of a mass storage device that describes the layout of the data on that device in terms of file names, length and location.  |
| Disk                         | A mass storage device. Basic unit is an electromagnetic platter on which data is magnetically recorded. Features random access and faster access time than magnetic tape.   |



|                        |  |
|------------------------|--|
| Display unit           | A device that provides a visual representation of data.  |
| Double buffered I/O    | An input (or output) operation using two buffers to achieve overlap. While one buffer is being used by the CPU program, the other is being read from (or written to) by an I/O device.   |
| Dummy file             | A file used only to hold space on a storage device.  |
| Editor                 | A program which interacts with the programmer to enter new programs into the computer and edit them as well as modify existing programs. Editors are language independent and will edit anything in alphanumeric representation. |
| EMT                    | A PDP-11 machine instruction (operation codes from 104000-104377). EMTs are most often used for monitor communication.   |
| Entry point            | A point in a subroutine to which control is transferred when the subroutine is called.   |
| Entry point table      | A table, kept by the Librarian program, of the names and locations of the routines available in the library.   |
| Entry symbol           | A global symbol defined in an object module that can be referred to by other object modules.   |
| Error flag             | Condition indicating an error has occurred. If the C bit is set on return from an RT-11 call, an error occurred. Thus, the C bit set is the RT-11 error flag.  |
| Exception              | An unusual condition (e.g., a floating point exception is an overflow).  |
| Expressions            | A combination of variables, constants, and operators (as in a mathematical expression).  |
| External symbol        | A symbolic name which is defined in one assembly or compilation and can be referenced in another. The .GLOBL directive is used to indicate external symbols to the MACRO assembler.  |
| Fatal error            | An error which makes continued processing impossible (e.g., running out of symbol table space is usually a fatal error).   |
| File allocation scheme | The method used to store data on I/O devices. RT-11 uses a contiguous structure.   |
| File gap               | A fixed length of blank tape separating files on a recording medium.   |



|                         |   |
|-------------------------|---|
| File structured device  | A device on which data is given names and arranged in files; the device also contains a directory of these names.   |
| Filler characters       | Null characters output to a device to give it time to perform unusually long operations (such as return the carriage) without explicitly waiting.   |
| Flag                    | A variable or register used to record the status of a program or device. In the latter case it is sometimes called a device flag.   |
| Floating-point          | A number system in which the position of the radix point is indicated by one part (the exponent part), and another part represents the significant digits, the fractional part (e.g., $5.39 \times 10^8$ - Decimal; $137.3 \times 8^4$ - Octal; $101.10 \times 2^{13}$ - Binary). |
| Foreground program      | A program of high priority that utilizes machine facilities when and as needed, but allows less critical background work to be performed in otherwise unused time.  |
| Formatted binary block  | A data structure used to hold binary information, usually to be read or written to a data file. For example, an object module produced by the MACRO assembler consists of formatted binary blocks.  |
| Fragmented              | Having many empty entries scattered over a device.  |
| Free blocks             | Areas of a file structured device which are unused.   |
| General registers       | A set of eight general purpose registers available for use as accumulators, as auto index registers or as pointers. General registers 6 and 7 serve as the hardware stack pointer and program counter respectively.   |
| Global symbol           | Any symbol accessible to other programs. Linkage must be supplied by a linker.  |
| Global symbol directory | The portion of an object module which describes all external symbols and CSECT names which are defined or referenced by the object module.  |
| Handler                 | see Device handler  |
| Hardware mode           | For magnetic tapes and cassettes. Allows full user control over position and record size. (As opposed to software mode.)  |
| High level language     | A language in which single statements may result in more than one machine language instruction, e.g., BASIC, FORTRAN, COBOL.  |



|                           |  |
|---------------------------|--|
| Image mode                | A mode of data transfer in PIP in which a file is copied without change of any kind.   |
| Implied argument          | An argument which is assumed by the program, whether or not it is explicitly stated by the user.   |
| Indexing                  | Using a variable index register as an offset into a table.   |
| Initialize                | To set counters, switches, addresses and variables to zero or other starting values.   |
| Internal symbol           | A symbolic name which is known only within the assembly or compilation where it is defined. Symbols are internal by default.   |
| Interrupt service routine | Routine entered when an external interrupt occurs.   |
| Interrupt                 | <ol style="list-style-type: none"> <li>1. To break the normal operation of the routine being executed and pass control to a specific location - generally accompanied by saving the state of the current routine so that control can return later.</li> <li>2. The signal which causes the break.</li> </ol> |
| Iterations                | Repetitions of a portion of a program.   |
| Job status word           | A word in the RT-11 communications region containing bit flags indicating the status of the program currently in memory.   |
| Keyboard monitor (KMON)   | Program that provides communication between the user at the console and the RT-11 system.  |
| Label                     | One or more characters used to identify a source language statement or line.   |
| LDA format file           | see Load image file  |
| Library                   | A collection of standard routines which can be incorporated into other programs.   |
| Library header            | Section of code that contains the current status of the library, including version, date and time of creation or update, relative starting address of entry point table, number of EPT entries and placing of next module to be inserted into the library file.  |
| Light pen                 | A device resembling a pencil or stylus which can detect a fluorescing spot on a CRT screen. Used to input information to a CRT display system.   |



|                  |   |
|------------------|---|
| Line-oriented    | Referring to editing operations on an entire line of text. See also Character-oriented.   |
| Load image file  | A program that can be executed in stand-alone environment without the aid of relocation.  |
| Local symbol     | A symbol used only within the program in which it is defined (all non-global symbols).  |
| Location counter | A counter kept by an assembler to determine the address assigned to an instruction or constant being assembled.   |
| Logical name     | A user-defined name assigned as an alternate name for a physical device. Useful for redirecting I/O when device originally specified is unavailable.  |
| Loop             | A sequence of instructions that is executed repeatedly until a terminal condition occurs. Also, used as a verb meaning to execute this circle of instructions while waiting for the ending condition. |
| Macro            | An instruction in a source language that is equivalent to a specified sequence of machine instructions, or a command in a command language that is equivalent to a specified sequence of commands.    |
| Macro symbol     | Symbol used as macro name in operator field.  |
| Mainstream code  | Any code which is not executing as a result of a completion routine.  |
| Mask word        | A combination of bits that is used to clear selected portions of any word, character, byte, or register while retaining other parts for use. Also, to clear these selected locations with a mask.     |
| Memory address   | An address in memory used to store data.  |
| Memory image     | An exact copy of a portion of memory. RT-11 save image files (.SAV) are memory images.  |
| Mnemonic         | Alphabetic representation of a function or octal machine instruction.   |
| Mode             | A state or method of program operation. Command mode causes user input to be interpreted as a command; text mode causes user input to be interpreted as alphanumeric data, etc.                       |
| Module           | A routine which handles a particular function.  |



|                            |  |
|----------------------------|--|
| Monitor                    | The collection of routines which schedules resources, I/O, system programs, and user programs, and obeys keyboard commands in a Monitor System.  |
| Monitor System             | Editors, assemblers, compilers, loaders, interpreters, data management programs and other utility programs all automated for the user by a monitor.  |
| Multi-region               | A term describing the RT-11 overlay structure in which there may exist multiple independent areas of high-speed memory (regions) in which overlays may occur.  |
| Nesting                    | <ol style="list-style-type: none"> <li>1. Including a program loop inside another program loop, or other similar occurrences within one another.</li> <li>2. Algebraic nesting such as <math>(A+B+(C+D))</math> where execution proceeds from the innermost level to the outermost level.</li> </ol> |
| Non-file structured device | A device used only for input or output, and not storage, of a file (e.g., line printer, or card reader) or which is a sequential device.   |
| Null                       | Characters with ASCII code 000.  |
| Object code                | The result after assembling or compiling source code. Machine code.  |
| Off-line                   | Pertaining to equipment, devices or events which are not under direct control of the computer.   |
| Offset                     | The difference between a location of interest and some known base location.  |
| On-line                    | Pertaining to equipment, devices and events which are in direct communication with the CPU and thereby under its control in some way.  |
| Open location              | Location whose contents have been printed for examination; contents can be changed.  |
| Operand                    | <ol style="list-style-type: none"> <li>1. A quantity which is affected, manipulated or operated upon.</li> <li>2. The contents of the field following the operator of an assembler instruction.</li> </ol>   |
| Operating system           | A program for automating a programmer's use of software. A Monitor System.   |
| Operator                   | That symbol or code which indicates an action or operation to be performed.  |



|                    |   |
|--------------------|---|
| Overlay            | The technique of repeatedly using the same area of memory during different stages of a program. When one routine is no longer needed in memory, another routine can replace all or part of it. Overlaying replaces parts of programs; chaining replaces the whole program.  |
| Overlay segment    | A section of code handled as a unit. This segment of code can overlay code already in memory or be overlaid by another overlay segment.   |
| Overlay-structured | A term describing a program which does not entirely reside in high-speed memory at any instant. Portions of the program are brought into memory when needed.  |
| Page of text       | That portion of a file delimited by form feed characters, generally 50-60 lines long; corresponds approximately to a physical page of program listing.  |
| Parameter          | A variable or an arbitrary constant appearing in a mathematical expression, each value of which restricts or determines the specific form of the expression.  |
| Patch              | To modify a routine in a rough or expedient way, usually by modifying the binary code rather than reassembling it.  |
| Peripheral devices | In a data processing system, any device distinct from the central processing unit, which may provide the system with outside storage or communication.  |
| Permanent file     | An output file that is stored in memory for later use.  |
| Permanent symbol   | Instruction mnemonics, assembler directives, and macro directives incorporated in the assembler.  |
| Physical device    | An input, output, or storage device associated with the Central Processing Unit.  |
| Physical name      | A 2- or 3-character name identifying a physical device. The first two characters are alphabetic, the third character is numeric.  |
| PIC code           | Abbreviation for Position Independent Code. (Code which can operate properly wherever it may be loaded in memory).  |
| Pointer            | <ol style="list-style-type: none"> <li>1. A location containing the address to another location.</li> <li>2. In EDIT program, a moveable reference pointer normally located between the character most recently operated upon and the next character in the buffer; represents the current position of the Editor in the text.</li> </ol> |



|                            |  |
|----------------------------|--|
| Proceed count              | In a program loop, the number of times the breakpoint is to be encountered before suspension of program execution.   |
| Processor status register  | A register that indicates the current priority of the processor, the condition of the previous operation, and other basic control items.   |
| Program counter            | A register in the CPU that holds the address of the current instruction being executed plus two; in other words, it holds the address of the next instruction unless the current instruction causes a jump.  |
| Program sections           | see Control section  |
| Programmed requests        | Machine language instruction which is used to invoke a monitor service for the issuing program.  |
| Purge                      | Deactivate a channel, without taking any other action.   |
| Radix                      | The base of a number system, the number of digit symbols required by a number system.  |
| Real time                  | <ol style="list-style-type: none"> <li>1. Pertaining to the actual time during which a physical process takes place.</li> <li>2. Pertaining to the performance of computer activity which occurs fast enough to influence the related physical process.</li> </ol> |
| Real time system           | A system in which computation is performed while a related physical process is occurring so that the results of the computation can be used in guiding the physical process.   |
| Record                     | A collection of related items of data treated as a unit. Example: A line of source code.   |
| Reentry address            | The Start address -2. Allows user program to reset itself internally, and resume operation.  |
| Region number              | A number which is used to identify a portion of memory to the Linker for the purpose of describing an overlay structure.   |
| Relocatable code           | Code written so that it can be located and executed in any part of memory, once it has been linked by Linker.  |
| Relocatable object modules | A set of instructions which are written so that the module can be loaded and executed in any memory area.  |



|                         |  |
|-------------------------|--|
| Relocatable symbols     | Symbolic names whose associated value is an offset from the base (beginning) of a control section. Such a symbol's value depends on the address of the control section and must be relocated each time the control section is assigned an address by the Linker. |
| Relocation              | To move a routine from one portion of storage to another and to adjust the necessary address references so that the routine can be executed in the new location.   |
| Relocation directory    | A portion of an object module which describes and identifies all occurrences of relocatable symbols in the object module.  |
| Repeat block            | Block of code to be repeated a defined number of times.  |
| Repeat count            | The number of times a block of code is to be repeated.   |
| Resident                | In memory, as opposed to being stored externally.  |
| Resident monitor (RMON) | The permanent memory-resident part of RT-11. Contains console terminal service, the error processor, the system device handler, the EMT processor, and system tables.  |
| Root segment            | See Base segment.  |
| Scratch area            | Any memory or registers used for temporary storage of partial results.   |
| Scrolling               | On a display screen, when the maximum number of lines are on the screen, the top line is deleted, and all text moves up one line, permitting one new line of display at the bottom of the screen. The visual effect is similar to the rolling up of a scroll.    |
| Sentinel file           | Last file on cassette tape; contains only header record; represents logical end-of-tape.   |
| Sequence number         | The number in a cassette directory which designates where a file is placed (in sequence), when that file has been continued on more than one cassette.   |
| Sequential              | A means of accessing data, in which records are read one after another from the device. As opposed to random access.   |
| Software mode           | For magnetic tapes and cassettes, the mode of operation when device access is through any RT-11 system program.  |



|                    |   |
|--------------------|---|
| Source code        | Text, usually in the form of an ASCII formatted file, which represents a program. Such a file can be processed by an appropriate system program (MACRO or FORTRAN) to produce an object module.   |
| Source file        | A file to be used as input to a translating program such as MACRO or BASIC.   |
| Stack              | An area of memory set aside by the programmer for temporary storage or subroutine interrupt service linkage. The stack uses the "last-in, first-out" concept. The stack starts at the highest location reserved for it and expands linearly downward as items are added to the stack.   |
| Stack pointer      | A general register used to keep track of the last location where data is entered into the stack.  |
| Storage device     | A general term for any device capable of retaining information.   |
| Subconditionals    | Directives that indicate: <ol style="list-style-type: none"> <li>1. assembly of an alternate body of code;</li> <li>2. assembly of a non-contiguous body of code within a conditional block;</li> <li>3. Unconditional assembly of a body of code within a conditional block.</li> </ol>  |
| Subpicture address | Address of a set of display processor instructions which display text on a GT40 (42,44).  |
| Suspend            | To temporarily halt execution of a task while another task of different priority runs.  |
| Swap               | The movement by the monitor of user programs between memory while they are running and a buffer area on a mass storage device when something else is running in that place in memory.   |
| Switch register    | A location in the CPU which can be loaded with a value by the operator, by his setting switches on the computer console for each bit he wants to enter.   |
| Switches           | <ol style="list-style-type: none"> <li>1. In a command string to the CSI a switch is a slash followed by an ASCII character which can be given a value by typing a colon after the character followed by an octal number or from 1-3 ASCII characters.</li> <li>2. Two or three position mechanisms used for operating computers or devices.</li> </ol> |



|                      |   |
|----------------------|---|
| Symbols              | Names which can be assigned values or which can be used to indicate specific locations in a program.  |
| Symbol table         | An array which contains all defined symbols and the binary value associated with each one. Mnemonic operators, labels and user defined symbols are all placed in the symbol table. (Mnemonic operators stay in the table permanently.)                                      |
| System configuration | see Configuration   |
| System device        | A peripheral mass storage device on which the Monitor resides.  |
| System programs      | DEC-supplied programs which come in the basic software packages. These include editors, PIP, assemblers, compilers, loaders, etc.   |
| Time-critical job    | A job which demands response within a fixed time period.  |
| Transfer address     | Program entry point.  |
| Trap                 | An automatic transfer of control to a prespecified routine that can be caused by software or hardware. The trap instruction is an example of a hardware implementation.   |
|                      | A conditional jump to a known location performed automatically by hardware as a side effect to executing a processor instruction. The address location from which the jump is made is recorded. It is distinguished from an interrupt which is caused by an external event. |
| Truncation           | The reduction of precision by ignoring one or more of the digits; not rounding off.   |
| Two's complement     | A number used to represent the negative of a given value in many computers. This number is formed from the given binary value by changing all 1s to 0s and all 0s to 1s, and then adding 1.   |
| Type-ahead           | The ability to type information at the console terminal and have it remembered by the system for later use.   |
| Unary operation      | Operation affecting a single operand. Examples: negation, radix indicator.  |
| User-defined symbol  | 1. A label<br>2. A symbol defined by direct assignment.   |
| Utility              | Any program which performs useful functions, i.e., PIP.   |



|                        |  |
|------------------------|--|
| Vector                 | Two words describing 1) where to go when external interrupt occurs, and 2) the contents of the processor status word when the interrupt is acknowledged. |
| Wild card operation    | A shorthand method of transferring all files with the same name, extension, or both.   |
| Word-for-word transfer | A transfer in which no alteration of data is performed.  |
| Words                  | In the PDP-11 a 16-bit unit of data which may be stored in an addressable location.  |
| Write-lock condition   | The condition of a device that is protected against any transfers which would write information to it.   |
| Zero                   | To initialize a device (e.g., DECtape, cassette) so that it contains no information.   |



## INDEX

- Absolute, 5-3
  - and relocatable program sections, 6-4
  - block numbers, 4-21, I-2
  - expression, 5-18, C-5
  - load address, 6-1
  - load module, 6-5
  - mode, 5-24
  - quantities, 5-17
  - section, 6-4
  - starting block, 4-17
- Absolute Loader, 6-5, A-8
- Accessing
  - general registers, 8-9
  - internal registers, 8-10
- Additional reference manuals, xvi
- Address mode syntax, C-5
- Address specifier, M-2
- Addressed location, 8-8
- Addresses, vector, 9-6
- Addressing modes, 5-20
- Advance command, 3-17
- Allocating,
  - blocks for files, 4-11
  - extra words, 4-18
  - memory for a queue, 9-66
  - system resources, 2-16
  - words, 2-35
- Alphabetize switch, 6-18
- Alphabetized load map, 6-19
- Alphanumeric representation, 1-2
- ALTMODE, 3-2
- Argument, 3-4
  - block, 9-3
  - CSECT, M-3
  - dummy, 5-67
  - iteration, 3-8
  - list, 9-5
  - list pointer, 9-2
  - negative line, 3-7
  - numeric, 3-6
  - numerical, 9-4
  - positive, 3-7
  - substitution, 5-72
- Arguments, 9-2
  - EDIT, 3-4
  - indefinite repeat, 5-72
  - missing, 5-66
  - number of, 5-66
  - real, 5-63
  - symbolic, 5-37
  - to Macro calls and definitions, 5-62
- ASCII,
  - character set, C-1, C-4
  - conversion of one or two characters, 5-40
  - files, 3-1
  - format, 2-3
  - input and output, 8-20
  - .ASCII directive, 5-41
  - .ASCIZ directive, 5-42
  - .ASECT directive, 5-53, 6-4, 6-6
  - .ASECTS, H-3
- ASEMBL (8K assembler), 1-4, 11-1
  - assembling and linking, A-18
  - calling and using, 11-1
  - error messages, 11-6
  - file specifications, 11-2
- Assembler, 1-2, 8-4
  - ASEMBL, 11-1
  - MACRO, 5-1
  - output, 5-19
- Assembler directives, 5-3, 5-27, C-19
  - .ASCII, 5-41
  - .ASCIZ, 5-42
  - .ASECT, 5-53
  - .BLKB, 5-47
  - .BLKW, 5-47
  - .BYTE, 5-38
  - .CSECT, 5-53
  - .DSABL, 5-36
  - .ENABL, 5-36
  - .END, 5-50
  - .ENDM, 5-60
  - .EOT, 5-51
  - .ERROR, 5-70
  - .EVEN, 5-46
  - .FLT2, 5-48
  - .FLT4, 5-48
  - .GLOBL, 5-54
  - .IDENT, 5-36
  - .IFF, 5-57
  - .IFT, 5-57
  - .IFTF, 5-57
  - .IRP, 5-71
  - .IRPC, 5-72
  - .LIMIT, 5-51
  - .LIST, 5-27
  - .MACRO, 5-60
  - .MCALL, 5-74
  - .MEXIT, 5-61
  - .NARG, 5-68
  - .NCHR, 5-68
  - .NLIST, 5-27
  - .NTYPE, 5-69



- .NTYPE, 5-69
- .ODD, 5-46
- .PAGE, 5-36
- .PRINT, 5-71
- .RADIX, 5-44
- .RAD50, 5-43
- .REPT, 5-73
- .SBTTL, 5-34
- .TITLE, 5-34
- .WORD, 5-39
- Assembly and Linking Instructions, A-14
  - ASEMBL, A-18
  - CREF, A-18
  - DUMP, A-20
  - EDIT, A-17
  - EXPAND, A-17
  - FILEX, A-19
  - LIBR, A-19
  - LINK, A-18
  - MACRO, A-17
  - ODT, A-21
  - PATCH, A-20
  - PIP, A-19
  - SRCCOM, A-20
  - the system files, A-15
  - the VT-11 Display Handler Library, A-21
- Assembling graphics programs, N-16
- Assembly instructions, N-24
- Assembly,
  - language display support, N-1
  - language statement, 5-2
  - link, and build instructions, A-1
  - listing, 8-1
  - listing table of contents, 5-35
  - location counter, 5-14
  - pass, 5-14
  - source listing showing local symbol blocks, 5-15
- ASSIGN command, 2-18
- Asterisk, wild-card, 4-1
- Asynchronous completion routines, 9-13
- Autodecrement mode, 5-23
- Autodecrement deferred mode, 5-23
- Autoincrement mode, 5-21
- Autoincrement deferred mode, 5-22
- Automatic generation of local symbols, 5-13
- Automatic relocation facility, 8-4
- Automatically created symbols within user-defined Macros, 5-66
- Backslash, 8-7
- Backup storage device, 6-10
- Back-arrow, 8-8
- Back space, H-11
- Bad block files, 4-2
- Bad block scan, 4-21
- Bad entry, 8-2, 8-26
- Base address, 5-25, 8-4
- Base command, 2-28
- Base segment, N-2
- BASIC/GT subroutine structure, N-20
- BASIC/RT-11, 1-1, 1-5
  - commands, F-3
  - error messages, F-6
  - function errors, F-9
  - functions, F-5
  - language, F-1
  - language summary, F-1
  - statements, F-1
  - string functions, F-5
- Beginning command, 3-16
- Binary,
  - code, 1-2
  - object module, 8-4
  - operators, 5-8, 5-18
  - output, 1-2
  - radix, 5-17, 5-45
- Bit patterns, 1-3, 8-24
  - search, 8-10
- .BLANK, N-3
- Blank COMMON, 6-14
- Blank,
  - extension filename, 2-7
  - fields, 9-110
  - lines, 5-2
- .BLKB directive, 5-47
- .BLKW directive, 5-47
- Blocks, 2-10
  - control, 6-1
  - device, 9-5
  - EMT arguments, 9-5
  - lengths, 4-4
  - 256-word, 2-31
- Block numbers, H-9
  - absolute, I-2
  - physical, I-2
- Block-replaceable devices, 2-35, 4-14
- BM792-YB hardware bootstrap, 2-1
- Boot operation, 4-20
- Bootstrap,
  - copy operation, 4-20
  - file, 4-19
  - hardware (cassette), A-6
  - loader, A-8
  - manual, 2-2
- Bootstrapping the system, 4-20
- Bottom address switch, 6-18
- Branch,
  - address, 5-26
  - instruction addressing, 5-26
  - instructions, 5-27, C-13
- Breakpoints, 8-11, 8-21
  - table, 8-10, 8-13
- Buffer,
  - flag, N-7, N-11
  - macro, 3-10
  - save, 3-10
  - structure, N-7
  - text, 3-2, 3-21



- Building,
  - a memory image, 2-28
  - DEctape from DEctape, A-2
  - DEctape from Disk, A-2
  - Disk from DEctape, A-2
  - Disk from Disk, A-2
  - Disk from Cassette, A-6
  - Disk from Paper Tape, A-8
  - RT-11 systems, A-1
  - VTLIB.OBJ, N-26
- Byte, 8-7
  - offset, 5-27
- BYTE command, M-3
- .BYTE directive, 5-38
- Calculating offsets, 8-17
- Calling and using,
  - ASSEMBL, 11-1
  - DUMP, I-1
  - EDIT, 3-1
  - EXPAND, 10-2
  - FILEX, J-1
  - LIBR, 7-1
  - LINK, 6-2
  - MACRO, 5-74
  - ODT, 8-1
  - PATCH, L-1
  - PATCHO, M-1
  - PIP, 4-1
  - SRCCOM, K-1
- Calls or branches to overlay
  - segments, 6-13
- Card codes, 2-24, H-6
  - conversion table, H-25
- Card reader handler, H-6
- Carry bit, 9-13
- Cassette,
  - rewind button, 4-7
  - sequence number, 4-7
- Cassette special functions, H-12
  - last block, H-12
  - last file, H-12
  - next block, H-12
  - next file, H-12
  - rewind, H-12
  - write file gap, H-13
- Cassettes and magtapes,
  - initializing, 4-13
  - legal operations, 4-5
- Cathode ray tube, N-1
- CBOOT, A-6
  - instructions, A-7
- CBUILD.SYS, A-6
- .CDFN request, 9-25
- Centralized queue management
  - system, 9-65
- CHAIN bit, 9-8
- .CHAIN request, 9-27
- Change command, 3-22
- Changing
  - device handler characteristics, 2-23
  - stack size, 2-35
- Channel, 9-41
  - data, 9-78
  - number, 9-5
  - status word, 9-13, 9-41, 9-78
- Channels, 9-35
- Character
  - deletion, 9-94
  - substitution, 5-72
  - transfer, 9-93
- Character set, 5-5
  - ASCII, C-4
  - Radix-50, C-3
- Character- and line-oriented
  - command properties, 3-6
- Character-oriented commands, 3-6
- Characters,
  - filler, A-11
  - illegal, 5-7
  - legal, 5-5
  - operator, 5-8
  - optional, 2-14
  - prompting, 2-4
  - special, 5-64
- .CHCOPY request, 9-28
- Checking channel status, 9-99
- Checksum, 4-9, M-4, M-5
- .CLEAR, N-4
- Clearing breakpoints, restarting
  - ODT, 8-2
- Clock, 2-17
  - frequency, 9-52
  - ticks, 9-51
- CLOSE command, 2-20
- CLOSE handler function, H-9
- .CLOSE request, 9-30
- Closed location, 8-6
- .CMKT request, 9-31
- .CNTXSW request, 9-32
- Code,
  - binary, 1-2
  - modifications, L-1
  - object, 1-2
  - source, 1-2
- Combining
  - files, 4-9
  - library switch functions, 7-11
- Command and Switch Summaries, B-1
- Command,
  - arguments (EDIT), 3-5, B-5
  - continuation switch, 7-3
  - decoder, 8-20
  - execution routine, 8-20
  - interpretation services, 9-1
  - mode, 3-2
  - repetition, 3-8
  - string format, 2-10
  - strings, 3-5, 6-2
  - structure (EDIT), 3-3



- switches, 4-1, K-2
- syntax (LIBR), 7-2
- Command String Interpreter, 2-7, 2-10, 6-3, 9-33, L-2
- Command summary,
  - ODT, B-12
  - PATCH, B-16
  - PATCHO, B-17
- Commands,
  - and functions, 8-5
  - BASIC/RT-11, F-3
  - character-oriented, 3-6
  - control, 3-2
  - display editor, 3-4
  - edit control, 3-2
  - editing, 3-10
  - input/output, 3-3, 3-10
  - keyboard, 2-14
  - line-oriented, 3-6
  - PATCH, L-2
  - pointer relocation, 3-16
  - search, 3-18
  - text modification, 3-4, 3-20
  - utility, 3-4, 3-24
- Commands to allocate system resources, 2-16
  - ASSIGN, 2-18
  - CLOSE, 2-20
  - DATE, 2-16
  - INITIALIZE, 2-18
  - LOAD, 2-20
  - SET, 2-23
  - TIME, 2-17
  - UNLOAD, 2-23
- Commands to control terminal I/O, 2-15
  - GT OFF, 2-16
  - GT ON, 2-15
- Commands to manipulate memory
  - images, 2-27
  - Base, 2-27
  - Deposit, 2-29
  - Examine, 2-29
  - GET, 2-27
  - SAVE, 2-30
- Commands used only in a F/B environment, 2-34
  - FRUN, 2-35
  - RSUME, 2-37
  - SUSPEND, 2-36
- Command field, 5-2, 5-4
- Common blocks, 6-14
- Compiling and linking PATCHO, A-20
- Completion functions, 9-14
- Completion routines, 2-36, 2-37, 9-13, 9-49, 9-60
- Component sizes, 2-9
- Components,
  - system hardware, 1-5
  - system software, 1-3
- Compress operation, 4-19
- Compressing
  - directories, 4-19
  - files, 4-19
- Concatenation, 5-67
- Condition codes, 8-10
- Conditional,
  - assembly directives, 5-55
  - block, 5-55
- Configuration word, 9-11, 9-52, 9-85
- Confirming file transfers, 4-10
- Console,
  - normal mode, 9-94
  - special mode, 9-94
  - terminal, 3-1, 3-27
  - terminal control and status registers, 9-12
- Constant register, 8-5, 8-10, 8-16
- Context switch, 9-32
- Contiguous,
  - area, 9-13
  - file, 7-12, 9-13, J-7
- Continuation lines, 5-2
- Continue switch, 6-20
- Control,
  - block, 6-1
  - commands, 3-2
  - parameters, 6-6
  - section names, 6-1
- Control section,
  - named, 6-4
  - unnamed, 6-4
- Controlling terminal I/O, 2-15
- Conventions, system, 2-3
- Converting V1 Macro calls to V2, 9-108
- Copy operations, 4-9
  - errors, 4-9
  - multiple, 4-11
- Copying,
  - files with the current data, 4-10
  - system files, 4-10
- Co-resident overlay routines, 6-22
- Core memory, 1-1
- Correct and incorrect macro calls, 9-3
- Correcting and updating object modules, M-1
- CR (card reader), H-6
- Creating,
  - a library file, 7-4
  - a new file, 4-6
- CREF, 1-4
  - assembling and linking, A-18
  - error messages, 5-86
  - listing output, 5-81, 5-82, 5-83
  - specification switches, 5-76
  - switches, 5-78, C-24
- Cross reference,
  - control sections, 5-78
  - errors, 5-78
  - listings, 1-4
  - MACRO symbols, 5-78



- permanent symbols, 5-78
- register-equate symbols, 5-78
- table generation, 5-78
- CR.SYS, A-3
- CSECT argument, M-3
- .CSECT directive, 5-53, 6-4
- CSECT, 7-13
- CSI,
  - error messages, 9-35
  - general mode, 9-34
  - special mode, 9-36
  - switch separators, 9-38
- .CSIGEN request, 9-33
- .CSISPC request, 9-36
- .CSTAT request, 9-41
- CTRL A, 2-12
- CTRL B, 2-12
- CTRL C, 2-12, 3-2, 8-3, H-5
- CTRL E, 2-12
- CTRL F, 2-12
- CTRL O, 2-13, 3-3, H-5
- CTRL Q, 2-13
- CTRL S, 2-13
- CTRL U, 2-13, 3-3, 8-4
- CTRL X, 3-3
- CTRL Z, 2-13, H-5
- CT.SYS, A-3
- Current,
  - location counter, 5-3, 5-14
  - location pointer, 3-6
  - subpicture tag, N-14
- Cursor, 3-28, N-2
- Customization for special hardware,
  - A-11
- Data,
  - format, 2-3, J-1
  - length, 9-78
  - storage directives, 5-37
  - transfer requests, 9-15
- DATE command, 2-16
- .DATE request, 9-20
- De-activating a channel, 9-65
- Debugger, 1-2
- Debugging,
  - process, 1-2
  - tool, 1-3
- DEC command, M-4
- DEC 026/029 card conversion table,
  - H-25
- Decimal,
  - number, 5-17
  - radix, 5-17, 5-45
- DECsystem-10,
  - DEctape, J-1
  - file-structured device, J-3
- DEctape,
  - DECsystem-10, J-1
  - PDP-11 DOS/BATCH, J-1
  - RSTS-11, J-1
- Default,
  - extensions, 9-34
  - FORTTRAN library switch, 6-20
  - stack, 9-7
- Defining NOTAG, N-19
- Delete
  - command, 3-21
  - global switch, 7-7
  - operation, 4-13
  - switch, 7-6
- DELETE handler function, H-8
- .DELETE request, 9-42
- Deleted files,
  - on cassette, 4-6
  - on magtape, 4-6
- Deleting files, 4-13
  - from DOS/BATCH (RSTS-11) DECTapes, J-7
- Delimiting characters, 3-25
  - separating and, 5-6
- Deposit command, 2-29
- Description of graphics macros, N-3
- Destination device, 4-9
- Device,
  - block, 9-5
  - designation, 3-12
  - file-formatted, J-1
  - name, logical, 2-5
  - name, physical, 2-19, 9-5
  - non-file structured, 6-21
  - ownership, 2-21
  - random access, 4-8, 6-1
  - size, 9-46
  - specification, 2-11
- Device Handlers, 2-8, 9-50, H-4
  - differences between V1 and V2, H-4
  - foreground programming and, H-1
  - loading, 9-50
  - user-written, H-4
- Device-independent functions, 9-85
- Device names,
  - logical, 2-5
  - permanent, 2-5
  - physical, 2-4
- .DEVICE request, 9-44
- DHALT display halt instruction, N-14
- Differences between V1 and V2
  - device handlers, H-4
  - entry conditions, H-4
  - header words, H-4
  - interrupt handling, H-4
- Differences between V1 and V2 RT-11, xv
- Direct assignment statement, 5-10, 5-14
  - conventions, 5-11
  - format, 5-10
- Directives,
  - assembler, see assembler directives
  - conditional assembly, 5-55



- data storage, 5-37
- immediate conditional, 5-58
- listing control, 5-27
- Macro, 5-60
- not available in ASEMBL, 11-2
- PAL-11R, 5-59
- PAL-11S, 5-59
- program boundaries, 5-51
- program section, 5-51
- terminating, 5-50
- Directory,
  - access motion and code consolidation, 9-79
  - blocks, 4-19
  - initialization operation, 4-18
  - list operations, 4-15
  - listings of magtapes, 4-7
  - segments, 4-18
- Directories, compressing, 4-19
- Disk, DOS/BATCH, J-1
- Display,
  - extended instructions, N-13
  - file handler, N-1, N-2
  - file structure, N-17
  - handler macro calls, N-16
  - hardware, 3-29
  - interrupt vectors, N-5
  - processor, N-1
  - program counter, N-14
  - screen, 3-27
  - status register, N-14
  - status words, N-8
  - stop instruction, N-2, N-18
  - stop interrupt handler, N-14
  - X register, N-14
  - Y register, N-14
- Display editor, 3-27
  - format, 3-28
  - using the, 3-28
- Display processor, N-1
  - management, N-2
  - mnemonics, N-16, N-23
- DJSR subroutine call instruction, N-13
- DNAME load name register instruction, N-15
- DOS/BATCH disk, J-1
- Double ALTMODE, 3-28
- Double-buffered I/O, 9-105
- Double Operand Instructions, C-8
- Double Register-Destination, C-17
- DRET subroutine return instruction, N-14
- .DSABL directive, 5-36
- DSTAT display status instruction, N-14
- .DSTATUS request, 9-45
- DTMNFBSYS, A-2
- DTMNSJSYS, A-2
- DTSYS, A-3
- Dummy,
  - argument, 5-67
  - argument list, 5-60
  - file, 4-6
  - names, 4-14
- DUMP, 1-5, I-1
  - assembling and linking, A-20
  - calling and using, I-1
  - error messages, I-5
  - switches, I-2, B-14
- DUMP command, M-4
- Duplicate entry points, 7-8
- E command, L-3
- EC command, 3-29
- Edge flag, N-12
- Edit Backup command, 3-11
- Edit Console command, 3-29
- Edit Display command, 3-29
- Edit Read command, 3-10
- Edit Version command, 3-27
- Edit Write command, 3-11
- Editor (EDIT), 1-2, 3-1
  - arguments, 3-4, 3-5, B-5
  - assembling and linking, A-17
  - calling and using, 3-1
  - command structure, 3-3
  - control commands, 3-2
  - display, 3-27
  - editing commands, 3-10
  - error messages, 3-33
  - example, 3-32
  - immediate mode commands, 3-30, B-8
  - input/output commands, 3-10, B-5
  - key commands, 3-2, B-8
  - modes of operation, 3-2
  - pointer relocation commands, 3-16, B-6
  - search commands, 3-18, B-6
  - text modification, 3-20
  - utility commands, 3-24, B-7
- Effective address search, 8-15
- Empty entry, 9-13, 9-43
- EMT,
  - and TRAP addressing, 5-27
  - argument blocks, 9-5
  - error code, 9-8
  - instruction, 9-2
- .ENABL directive, 5-36
- .END directive, 5-50
- End File command, 3-15
- .ENDM directive, 5-60
- .ENDM statement, 5-74
- .ENDR statement, 5-74
- ENTER handler function, H-8
- .ENTER request, 9-47
- Entering I/O information, 2-10
- Entry conditions, H-4



- Entry point, 5-10, 5-55, 6-7, 6-13, 9-46
  - table, 7-5, 7-12, 7-13
- Entry symbol, 6-5
- .EOT directive, 5-51
- Error,
  - code, 9-35
  - returns, 9-99
  - types, P-1
- .ERROR directive, 5-70
- Error halt bit, 9-8
- Error messages,
  - ASSEMBL, 11-6
  - BASIC/RT-11, F-6
  - CREF, 5-86
  - DUMP, I-5
  - EDIT, 3-33
  - fatal monitor, 2-38
  - keyboard, 9-40
  - MACRO, 5-84
  - monitor, 2-37
  - PATCH, L-7
  - PATCHO, M-7
  - PIP, 4-24
  - summary, P-1
- Evaluation of an expression, 5-18
- Even byte, 9-6
- .EVEN directive, 5-46
- Examine, change locations in the file, L-3
- Examine command, 2-29
- Examples,
  - device handlers, H-14
  - EDIT, 3-32
  - MACRO line printer listing, 5-31
  - page heading, 5-32
  - PATCH, L-4
  - PATCHO, M-5
  - using GTON, N-28
- EX command, 3-16
- Exchange command, 3-23
- Execute Macro command, 3-26
- Exit command, 3-15
- EXIT command, M-4
- Exit from PATCH, L-3
- .EXIT request, 9-49
- EXPAND, 1-4, 10-1
  - assembling and linking, A-17
  - calling and using, 10-2
  - error messages, 10-6
  - language, 10-1
  - restrictions, 10-1
- Expression, C-5
  - absolute, 5-18, C-5
  - external, 5-18
  - register, 5-20, C-5
  - relocatable, 5-18, 8-4
- Expressions, 5-18
  - symbols and, 5-5
- Extend and delete operations, 4-13
- Extended display instructions, N-13
- Extending file lengths, 4-13
- Extensions, 3-12, K-2
  - and filenames, 2-5
- External,
  - expression, 5-18
  - symbols, 5-10, 6-5
- F command, L-2
- Facilities,
  - available only in RT-11 F/B, 1-7
  - for input and output operations, 9-1
- Fast listing, J-7
- Fatal monitor error messages, 2-38
- F/B Monitor, 9-7
- F/B programming and device handlers, H-1
- F/B programming in RT-11, Version 2, H-1
- F/B system, background area, 2-9
- .FETCH request, 9-50
- Field,
  - comment, 5-2, 5-4
  - label, 5-2, 5-3
  - operand, 5-4
- File,
  - allocation scheme, 4-10
  - ASCII, 3-1
  - descriptor blocks, 9-37
  - dummy, 4-6
  - formats, J-1
  - gap, 9-86, H-11
  - length, 9-78
  - manipulation requests, 9-15
  - manipulation services, 9-1
  - memory image, 2-3, 4-9, L-1
  - names and extensions, 2-5
  - non-overlay, L-3
  - overlay, L-1
  - permanent, 9-13
  - relocatable image, 2-3
  - replacement, 4-8
  - sentinel, 4-4
  - specifications (ASSEMBL), 11-2
  - structure, 9-13, H-6
  - temporary, 4-23
  - tentative, 9-13
  - transfer, 4-1
- File-formatted devices, J-1
- File-structured RT-11 device, J-1
- Filename, 3-12
  - blank extensions, 2-7
  - extensions, 2-6
  - input, 4-1
  - output, 4-1
- Files, compressing, 4-19
- FILEX, 1-4, J-1
  - assembling and linking, A-19
  - calling and using, J-1
  - error messages, J-8
  - overview, J-1
  - switch options, J-2, B-15



Fill characters, 2-31, 9-9, A-11  
 Fill count, 9-9  
 Find command, 3-19  
 Fixed record length, H-6  
 Floating point,  
   exception, 9-84  
   hardware, 5-47, 9-84  
   numbers, 5-17, 5-48  
   source double register, C-15  
 .FLT2 directive, 5-48  
 .FLT4 directive, 5-48  
 Foreground/Background, 1-1  
   terminal I/O, 2-13  
 Foreground links, 6-6  
 Format,  
   ASCII, 2-3  
   control, 5-5  
   load image, 2-3, 6-2, 6-21  
   memory image, 2-3  
   object, 2-3  
   of Entry Point Table, 7-13  
   of library files, 7-12  
   of programmed requests, 9-2  
   relocatable image, 2-3  
   register, 8-6  
   statement, 5-2  
 Formats, data, 2-3  
 Formatted binary copy switch, 4-9  
 Formatting,  
   horizontal, 5-5  
   vertical, 5-5  
 Form feed character, 3-1, 3-12  
 Forms of relocatable expressions,  
   8-5  
 FORTRAN IV, 1-1, 1-5, G-1  
   character set, G-2  
   compiler error diagnostics, G-11  
   expression operators, G-3  
   object time error diagnostics,  
     G-11  
   overlays, G-14  
   running a FORTRAN program in the  
     foreground, G-1  
   statements, G-4  
 Forward space, H-11  
 Fragmented device, 9-13  
 Free area, 4-11  
 Free memory list, 2-9, 2-22  
 FRUN command, 2-35  
 FRUN processor, 9-7  
 Function,  
   code, 9-6  
   control switches, C-23  
   switches, 5-76, 5-77  
 Functions, BASIC/RT-11, F-5  
  
 General,  
   address specification, 5-21  
   file transfer program, J-1  
   library file format, 7-12  
   memory layout, 2-8  
   registers, 5-11, 8-9  
  
 GET command, 2-27  
 Get command, 3-18  
 Global symbol directory, 7-14  
 Global symbol table, 6-1, 6-5  
 Global symbols, 5-10, 5-55, 6-1,  
   6-5, M-3  
 Globals, unresolved, 6-1  
 .GLOBL directive, 5-54  
 Graphics macro calls, summary, N-21  
 Graphics programs,  
   assembling and linking, N-16  
 GT OFF, 1-5, 2-15, 2-16, N-2  
 GT ON, 1-5, 2-15, 3-29, N-2  
 .GTIM request, 9-51  
 .GTJB request, 9-52  
  
 HALT instructions, 2-40  
 Handler functions, H-8  
   CLOSE, H-9  
   DELETE, H-8  
   ENTER, H-8  
   LOOKUP, H-8  
   READ/WRITE, H-9  
 Handler size, 9-46  
 Handlers, 2-35  
   device, 2-8, H-1  
   display file, N-2  
   internal display file, N-2  
   removing from memory, 9-74  
   system, A-1  
 Hardware bootstrap, 4-20  
   BM792-YB, 2-1  
   cassette, A-6  
   MR11-DB, 2-2  
 Hardware,  
   configuration, 2-1  
   display, 3-29  
   memory protection, 9-6  
   mode, H-6  
 Header words, H-4  
 HELP command, M-5  
 .HERR request, 9-53  
 High,  
   address, 2-31  
   baud rate serial console devices,  
     A-11  
   level languages, 1-3  
   memory address, 9-8  
   order time of day, 9-52  
 Horizontal formatting, 5-5  
 .HRESET request, 9-55  
  
 Identification messages, 2-3  
 .IFF Directive, 5-57  
 .IFT Directive, 5-57  
 .ITF Directive, 5-57  
 Illegal characters, 5-7, 8-26  
 Immediate conditional directive,  
   5-58



Immediate mode, 1-3, 3-2, 3-4,  
     3-30, 5-24, 9-3  
     commands, 3-30, B-8  
 Important memory areas, 9-6  
 Improving assembler performance,  
     A-14  
 Include switch, 6-20  
 Inclusive directory, 4-5  
 Indefinite repeat  
     arguments, 5-72  
     block, 5-71, 5-72  
     .IDENT directive, 5-36  
 Index Mode, 5-23, 5-25  
 Index Deferred Mode, 5-23  
 Individual module name, 7-2  
 Inhibit TT wait bit, 9-8  
 INITIALIZE command, 2-18, 3-30  
 Initializing cassettes and  
     magtapes, 4-13  
 Input and output, 3-3, 6-5  
     commands (Editor), 3-10, B-5  
 Input,  
     filenames, 4-1  
     list, 2-11  
     ring buffer, H-5  
     source filename, I-1  
 Insert command, 3-20  
 Inserting modules into a library,  
     7-5  
     .INSRT, N-5  
 Instruction,  
     EMT, 9-2  
     mnemonic, 5-3  
     offset, 8-17  
 Instructions, C-6  
     branch, C-13  
     double operand, C-8  
     operate, C-11  
     rotate/shift, C-9  
     single operand, C-8  
     trap, C-12  
     .INTEN request, 9-21, H-4  
 Internal  
     buffers, 3-1  
     Macro buffer, 3-26  
     registers, 8-10  
     subpicture stack, N-14  
     symbol directory, 7-14  
     symbolic names, 5-53  
     symbols, 5-10  
     tables, 10-1  
 Interrupt  
     enable bit, 9-44  
     handling, H-4, N-2  
     level (issuing programmed  
         requests), H-2  
     priorities, H-1  
     priority level, 8-10  
     Service Routine, H-2, H-3  
     vector, H-1, H-3  
 Invalid punch combinations,  
     H-6  
 I/O count, 9-78  
     exit routine, 9-11  
     vector, 9-33  
     .IRP and .IRPC example, 5-73  
     .IRP directive, 5-71  
     .IRPC directive, 5-72  
 Issuing programmed requests at the  
     interrupt level, H-2  
 Iteration argument, 3-8  
     loops, 3-9  
  
 Job Status Word, 2-31, 2-34, 9-7,  
     9-94  
 Jump command, 3-17  
 Jumps, 6-13  
  
 Key commands, Editor, 3-2, B-8  
 Key, LINE FEED, 8-7  
 Keyboard,  
     commands, 2-14  
     communication (KMON), 2-11  
     error messages, 9-40  
     monitor (KMON), 2-7  
 Keyboard Monitor (KMON), 2-7  
     command summary, B-1  
     special function keys, B-3  
 Kill command, 3-22  
  
 Label field, 5-2, 5-3  
 Language summary, BASIC/RT-11, F-1  
 Last block, H-12  
 Last file, H-12  
 LDA format, 6-2  
     switch, 6-21  
 Legal  
     argument delimiters, 5-63  
     characters, 5-5  
     operations involving cassette  
         or magtape, 4-5  
     separating characters, 5-6, 5-67  
     wild card, 4-2  
 Librarian (LIBR), 1-2, 1-3, 1-4, 7-1  
     assembling and linking, A-19  
     calling and using, 7-1  
     error messages, 7-14  
     switch commands, 7-2, 7-3, B-12  
 Library  
     directory, 7-1  
     end trailer, 7-14  
     header, 7-1, 7-12, 7-13  
     processing, 7-14  
     searches, 6-17  
 Library files, 6-8  
     creation, 7-4  
     directory listing, 7-9  
     entry point table, 7-7  
     format, 7-12  
     inserting modules into, 7-5  
     merging, 7-10



Light pen, N-1  
   status buffer, N-11  
 .LIMIT directive, 5-51  
 Limitations, PATCHO, M-5  
 Line- and character-oriented  
   command properties, 3-6  
 line deletion, 9-94  
   formatting, 5-5  
   oriented commands, 3-6  
   printer overstriking capability,  
     2-23  
 LINE FEED key, 8-7  
 Linker (LINK), 1-2, 1-4, 6-1, 7-14  
   assembling and linking, A-18  
   calling and using, 6-2  
   error handling and messages, 6-24  
   input and output, 6-5  
   load map, 8-2  
   load map for background job,  
     6-9  
   switches, 6-3, B-11  
 Linked  
   list, 9-66  
   program, 6-1  
 Linking graphics programs, N-16  
 Linking, relocation and, 5-19  
 List command, 3-14  
 LIST command, M-4  
 .LIST directive, 5-27  
 Listing,  
   assembly, 8-1  
   control directives, 5-27  
   control switches, 5-76, C-23  
   cross-reference, 1-4  
   directories, J-6  
   level count, 5-28  
   the directory of a library file,  
     7-9  
 .LNKRT, N-5  
 Load address, 2-35  
   absolute, 6-1  
 LOAD command, 2-20  
 Load image format (.LDA), 2-3  
 Load  
   map, 1-2, 6-1, 6-7  
   module, 1-2, 6-1, 6-5, 6-21  
 Loader,  
   absolute, A-8  
   bootstrap, A-8  
 Loading  
   device handlers, 9-50  
   device registers, 9-44  
   memory image files, 2-32  
   ODT with user program, 8-2  
   root segment, 2-28  
 Local symbol block, 5-13  
 Local symbols, 5-12, 5-66  
   automatic generation, 5-13  
 Location,  
   addressed, 8-8  
   closed, 8-6  
   counter control, 5-46  
   open, 8-6  
 Locations, opening, changing, and  
   closing, 8-6  
 .LOCK request, 9-56  
 Locking USR in memory, 9-56  
 Logical device name, 2-5  
   end-of-tape, 4-5  
   identifier, 9-5  
   names, 2-5, 2-18  
 LOOKUP handler function, H-8  
 .LOOKUP request, 9-58  
 Lowering the processor priority,  
   H-2  
 Low and high addresses, 5-51  
   -order priority, 8-19  
   priority, 8-19  
 .LPEN, N-7  
 LP.SYS, A-3  
 L-shaped cursor, 3-28  
 LSRA instruction, N-14  
  
 Machine language, 1-2  
 Macro,  
   arguments, 5-7  
   buffer, 3-10  
   call, 5-3, 9-2  
   call operator, 5-3  
   definition, 5-60  
   definition formatting, 5-61  
   expansion, 9-24  
   free source code, 10-1  
   libraries, 5-74  
   nesting, 5-63  
   recursive, 10-6  
   references, 10-1  
 Macro assembler, 1-1, 1-3, 5-1  
   assembling and linking, A-17  
   calling and using, 5-74  
   character code, C-1  
   directives, 5-60  
   error messages, 5-84  
   features, 5-1  
   instructions, C-1  
   program section capabilities, 5-10  
   source code, 5-80, 5-81  
   source statements, 5-2  
   special characters, C-5  
   switches, 5-76, C-23  
   symbol table, 5-9  
   symbols, 5-9  
 Macro calls,  
   requiring no conversion, 9-108  
   which may be converted, 9-108  
 Macro command, 3-10, 3-25  
 .MACRO directive, 5-60  
 Magtape, 7-track, A-11  
 Magtape special functions, H-10  
   back space, H-11  
   forward space, H-11  
   off line, H-12  
   rewind, H-10  
   write end-of-file, H-11  
   write with extended gap, H-11



- Main file/subroutine structure, N-19
- Main line code, H-5
- Mainstream code, 9-87
- Making patches permanent, 2-27
- Manipulating memory images, 2-27
- Manual bootstraps, 2-2
- Mark time, 1-7
  - requests, 9-31
- Mask limit, 8-15
- Masking, 8-10
- Matching areas, K-1
- Maximum file size, 9-12
- .MCALL directives, 5-74
- Memory block initialization, 8-16
- Memory
  - core, 1-1
  - diagram, BASIC link with overlay regions, 6-11
  - maps, 2-8, 8-4
  - solid state, 1-1
  - usage, 3-9
  - usage map, 6-6
- Memory image, L-1
  - files, 2-3, 4-9, L-1
  - format, 2-3
- Merging library files, 7-10
- .MEXIT directive, 5-61
- Minimum memory configurations, 5-1
- Miscellaneous services, 9-16
- Missing arguments, 5-66
- Mnemonics, 1-2
- Mode,
  - absolute, 5-24
  - autodecrement, 5-23
  - autodecrement deferred, 5-23
  - autoincrement, 5-21
  - autoincrement deferred, 5-22
  - command, 3-2
  - dump, A-11
  - general, 9-34
  - hardware, H-6
  - immediate, 1-3, 3-2, 3-4, 3-30, 5-24
  - index, 5-23, 5-25
  - index deferred, 5-23
  - instruction, 8-17
  - non-dump, A-11
  - Radix-50, 8-10
  - register, 5-21
  - register deferred, 5-21
  - relative, 5-24
  - relative branch, 8-9
  - relative deferred, 5-25
  - single instruction, 8-14, 8-17
  - software, H-6
  - source operand, 5-25
  - special, 9-36
  - text, 3-2, 3-20
  - type-ahead, 2-14
- Modes, addressing, 5-20
  - of operation (EDIT), 3-2
- Modify stack address, 6-21
- Modules, 1-2, 6-5
  - absolute load, 6-5
  - load, 6-1, 6-5, 6-21, 7-1
  - object, 6-5, 6-16, 7-1, 7-13, 7-14
- Monitor, 1-1, 2-1
  - error messages, 2-37
  - F/B; 1-6, 9-7
  - HALTs, 2-40
  - keyboard (KMON), 2-7
  - memory protection map, 6-6
  - Resident (RMON), 2-7, 9-7
  - running version, A-1
  - single-job, 1-6, 3-30, 9-7
  - software components, 2-7
  - start procedure, 2-1
  - version number, 9-11
- MONITR.SYS, A-1, A-2, A-6
- MOV instruction, 9-3
- MR11-DB hardware bootstrap, 2-2
- .MRKT request, 9-60
- MT/CT (Magtape (TU10/TM11) and Cassette (TA11)), H-6
  - general characteristics, H-6
- MT.SYS, A-3
- Multiple,
  - command lines, 2-14
  - copy operations, 4-11
  - delimiters, 5-4
  - GETs, 2-28
  - labels, 5-3
  - operands, 5-4, 5-38
  - .QSET requests, 9-66
- Multiply-defined symbols, 5-13
- .MWAIT request, 9-62
- [n] construction, 4-11
- .NAME, N-9
- Name register,
  - contents, N-14
  - internal software register, N-15
- .NAME request, N-9
- Name value, N-7
- Named
  - COMMON, 6-14
  - control sections, 6-4
  - relocatable program sections, 5-53, 5-54
- .NARG directive, 5-68
- .NCHR directive, 5-68
- Negative,
  - line arguments, 3-7
  - numbers, 5-17
- Nesting level, 5-6
- Next
  - block, H-12
  - file, H-12
- Next command, 3-14
- .NLIST, 5-27
- Non-dump mode, A-11



- Non-file structured,
  - delete, 4-5
  - devices, 6-21
  - lookup, H-6
- Non-overlay file, L-3
- Non-time-critical job, 1-7
- Nonexistent symbol, 6-21
- .NTYPE directive, 5-69
- Null specification, 11-2
- Number, C-18
  - channel, 9-5
  - decimal, 5-17
  - monitor version, 9-11
  - of arguments, 5-66
  - update, 9-11
- Numbers,
  - floating-point, 5-17, 5-48
  - MACRO assembler, 5-17
  - negative, 5-17
  - octal, 5-17
  - positive, 5-17
  - software identification, xiv
- Numeric arguments, 3-6, 9-4
  - passed as symbols, 5-64
- Numeric control, 5-47
- Object
  - code, 1-2
  - files, 6-5
  - format, 2-3
  - modules, 5-19, 6-5, 6-16, 7-1, 7-13, 7-14, A-8
  - modules, relocatable, 8-1
  - modules, starting point, 7-13
  - output, 1-2
- .OBJ format, M-1
- Octal,
  - channel number, 9-13
  - numbers, 5-17
  - radix, 5-17, 5-45
- .ODD directive, 5-46
- Odd (high-order) byte, 9-6
- ODT
  - (On-line debugging technique), 1-5
  - break routine, 8-23
  - command summary, B-12
  - error detection, 8-25
  - functional organization, 8-20
  - priority bit, 8-2
  - priority level, 8-19
- Offset words, 9-11
- OLDPIP, A-9
- On-line debugging techniques (ODT)
  - see ODT
- OPEN command, M-1
- Open file, 2-20
- Opening a byte address, L-3
- Opening, changing, and closing
  - locations, 8-6
- Operand field, 5-4
- Operands, multiple, 5-38
- Operate Instructions, C-11
- Operation,
  - foreground/background, 1-1
  - single-job, 1-1
- Operations on files, 4-2
  - magtape and cassette, 4-4
- Operator
  - characters, 5-8
  - field, 5-3
- Operators,
  - binary, 5-8, 5-18
  - unary, 5-8, 5-45, 5-50
- Optimizing device motion, A-14
- Optional
  - characters, 2-14
  - hardware devices, 1-5
- Output,
  - filenames, 4-1
  - format, K-2
  - list, 2-11
  - ring buffer, H-5
- Packed image transfer, J-2
- Page, 3-1
  - eject, 5-36, 5-61
  - headings, 5-34
- .PAGE directive, 5-36
- PAL-11R directive, 5-59
- PAL-11S conditional assembly
  - directive, 5-59
- Parameter list, 2-30
- Parameters used as arguments, E-1
- Passing switch information, 9-35
  - 9-38
- PATCH utility program, 1-4, L-1
  - commands, L-2
  - command summary, B-16
  - error messages, L-7
  - examples, L-4
- PATCH,
  - assembling and linking, A-20
  - calling and using, L-1
- Patching,
  - libraries, M-1
  - new files, L-2
  - OBJ files, M-1
- PATCHO, 1-4
  - commands, M-1
  - command summary, B-17
  - error messages, M-7
- Offline, H-12
- Offset, 5-13, 5-25
  - relative branch, 8-9
  - relative branch instruction, 8-17



- examples, M-5
- limitations, M-5
- run-time error messages, M-8
- PATCHO,
  - calling and using, M-1
  - compiling and linking, A-20
- PDP-11 DOS/BATCH DECTape, J-1
- Percent (%) character, 5-12
- Peripheral Interchange Program, (PIP), see PIP
- Permanent,
  - device names, 2-5
  - files, 2-20, 9-13
- Permanent Symbol Table, 5-9
- Physical block numbers, I-2
- Physical device, 2-19, 9-5
  - names, 2-4
- PIC (position independent code), 8-18, 9-21
- PIP (Peripheral Interchange Program), 1-4, 4-1
  - copy operations, 4-9
  - error messages, 4-24
  - magtape or cassette operations, 4-4
  - switches, 4-2, 4-3
  - switch summary, B-9
  - warning messages, 4-25
- PIP,
  - assembling and linking, A-19
  - calling and using, 4-1
- POINT Command, M-2
- Pointer location, 3-3
- Pointer relocation commands, 3-16
  - for Editor, B-6
- Position command, 3-20
- Position independent code, (PIC), 8-18, 9-21
- Positive,
  - arguments, 3-7, 4-5
  - numbers, 5-17
- Power line synchronization
  - feature, N-11
- PP.SYS, A-3
- PR (High-Speed Paper Tape Reader), H-5
- .PRINT directive, 5-71
- .PRINT request, 9-63
- Printout formats, 8-5
- Priorities, Interrupt, H-1
- Priority, C-18
- Proceed command, 8-12
- Proceed count, 8-13
- Processor Status, 8-1
- Program Boundaries directive, 5-51
- Program,
  - counter, 5-20, 8-8
  - development aids, 1-2
  - execution, 8-12
  - runaway, 8-23
  - section directives, 5-51
  - section names, 5-53
  - sections, absolute and relocatable, 6-4
- Program starting commands, 2-32
  - R, 2-33
  - REENTER, 2-34
  - RUN, 2-32
  - START, 2-33
- Programmed requests, 2-7, 9-1
  - format, 9-2
  - summary, E-1
  - usage, 9-25
- Programming
  - considerations, 8-20
  - conventions, H-1
  - errors, 1-3
- Prompting characters, 2-4
- .PROTECT request, 9-64, H-3
- PR.SYS, A-3
- .PURGE request, 9-65
- Purging an inactive channel, 9-65
- QCBOOT, A-6, A-7
- .QSET request, 9-65
- Quantities, absolute, 5-17
- Queue element, 9-61
- .RADIX directive, 5-44
- Radix,
  - binary, 5-17, 5-45
  - decimal, 5-17, 5-45
  - octal, 5-17, 5-45
- Radix,
  - control, 5-44
  - specification characters, 5-45
- Radix -50,
  - character set, C-3
  - equivalents, C-4
  - mode, 8-10
  - notation, 5-36
  - terminators, 8-11
- .RAD50 directive, 5-43
- Random access,
  - devices, 4-8, 6-1
  - file capabilities, F-1
- R Command, 2-33
- .RCTRLO request, 9-67
- .RCVD request, 9-68
- .RCVDC request, 9-69
- .RCVDW request, 9-69
- READ command, 3-12
- .READ request, 9-71
- .READC request, 9-72
- .READW request, 9-73
- READ/WRITE handler function, 9-9
- Real arguments, 5-63
- Re-assembling, 1-3
- Rebooting the system, 4-19
- Receiving data, 9-68
- Reclaiming memory, 2-22
- Reconfiguration, 1-5
- Record, H-9
- Recovering files, 4-14
- Recovery from Bad Blocks, 4-21



Recovery procedures during output operations, 4-8  
 Recurring coding sequence, 5-60  
 Recursive macros, 10-2  
 Reducing disk fragmentation, 4-12  
 Re-editing, 1-3  
 Reenter bit, 2-34, 9-7  
 REENTER command, 2-34, 3-2  
 Reference line, K-3  
 .REGDEF, macro call, 9-22  
 Region number, 6-13  
 Region, overlay, 6-10, 6-14  
 Register Deferred Mode, 5-21  
 Register,  
     destination, C-14  
     expression, 5-20, C-5  
     mnemonic, 9-3  
     mode, 5-21  
     symbols, 5-11, 5-12  
 Register-Offset, C-14  
 Registers,  
     console terminal control and status, 9-12  
     constant, 8-16  
     relocation, L-1  
 Reinitializing monitor tables, 4-20  
 Relative,  
     branch, 8-15  
     branch offset, 8-9  
     deferred mode, 5-25  
     mode, 5-24  
 .RELEASE request, 9-74  
 Releasing USR from memory, 9-57  
 REL format, 6-1  
     output file, 6-2  
     switch, 6-23  
 Relocatable, 1-2, 5-3  
     expressions, 5-18, 8-4  
     image file, 2-3  
     image format (.REL), 2-3  
     object module, 8-1  
 Relocation, 8-4  
     base, 2-28  
     bias, 8-4, 8-17  
     calculators, 8-18  
     constant, 5-3  
     directory, 7-14  
     register commands, 8-17  
     registers, 8-6, L-1  
 Relocation and Linking, 5-19  
 .REMOV, N-9  
 Removing,  
     handlers from memory, 9-74  
     logical assignments, 2-19  
     terminated jobs, 2-22  
 Rename operation, 4-15  
 .RENAME request, 9-75  
 .REOPEN request, 9-77  
 Repeat block, 5-73  
 Repeat counts, 8-23  
 Replace switch, 7-5  
 Replacement file, 4-8  
 .REPT directive, 5-73  
 Requests  
     for Data Transfer, 9-14  
     for File Manipulation, 9-14  
     for Miscellaneous Services, 9-14  
     requiring the USR, 9-19  
 Requests, programmed, 9-1  
 Requests,  
     .CDFN  
     .CHAIN, 9-27  
     .CHCOPY, 9-28  
     .CLOSE, 9-30  
     .CMKT, 9-31  
     .CNTXSW, 9-32  
     .CSIGEN, 9-33  
     .CSISPC, 9-36  
     .CSTAT, 9-41  
     .DELETE, 9-42  
     .DEVICE, 9-44  
     .DSTATUS, 9-45  
     .ENTER, 9-47  
     .EXIT, 9-49  
     .FETCH, 9-50  
     .GTIM, 9-51  
     .GTJB, 9-52  
     .HERR, 9-53  
     .HRESET, 9-55  
     .LOCK, 9-56  
     .LOOKUP, 9-58  
     .MRKT, 9-60  
     .MWAIT, 9-62  
     .PRINT, 9-63  
     .PROTECT, 9-64  
     .PURGE, 9-65  
     .QSET, 9-65  
     .RCTRLO, 9-67  
     .RCVD, 9-68  
     .RCVD, 9-69  
     .RCVDW, 9-69  
     .READ, 9-71  
     .READC, 9-72  
     .READW, 9-73  
     .RELEAS, 9-74  
     .RENAME, 9-75  
     .REOPEN, 9-77  
     .RSUM, 9-87  
     .SAVESTATUS, 9-77  
     .SDAT, 9-80  
     .SDATC, 9-81  
     .SDATW, 9-81  
     .SERR, 9-53  
     .SETTOP, 9-82  
     .SFPA, 9-84  
     .SPFUN, 9-85  
     .SPND, 9-87  
     .SRESET, 9-90  
     .TLOCK, 9-91  
     .TRPSET, 9-92  
     .TTINR, 9-94  
     .TTYIN, 9-93  
     .TTYOUT, 9-95  
     .TTOUTR, 9-95  
     .TWAIT, 9-98



.UNLOCK, 9-57  
 .WAIT, 9-71, 9-99  
 .WRITC, 9-101  
 .WRITE, 9-100  
 .WRITW, 9-102  
 Reserving storage area, 5-16  
 Resident monitor (RMON), 2-7, 9-7  
 Resident overlay handler, 6-15  
 Restarting ODT, clearing break-points, 8-2  
 Restarting PIP, 4-1  
 .RESTR, N-9  
 Restrictions on expanding macros, 10-1  
 Return  
   from Interrupt Service, H-2  
   to previous sequence, 8-9  
   to monitor, CTRL C, 8-3  
 Return path, 6-13  
 Reusing bad blocks, 4-19  
 Rewind, H-10, H-12  
 RFl1 platter, A-12  
 Ring buffer I/O, H-5  
 RFMNFBSYS, A-3  
 RF.SYS, A-3  
 RKMNFBSYS, A-2  
 RKMNSJSYS, A-2  
 RK.SYS, A-3  
 Root segment, 2-28, 6-10, 6-14, L-3  
 Rotate/Shift Instructions, C-9  
 Rounding numbers, 5-48  
 Routines, 1-3  
 Routines, completion, 2-36, 2-37  
 Routine, Service Interrupt, H-2  
 RSTS-11 DECTape, J-1  
 .SUM request, 9-87  
 RSUME Command, 2-37  
 RT-11 System, 1-1  
   data formats, file transfers, 4-1  
   DECTape control handler, J-5  
   Foreground/Background monitor, 1-6  
   I/O transfers, 9-65  
   librarian, 7-1  
   magtape handler, A-11  
   Memory Map (GT40), 2-9  
   operating environments, 1-1  
   Single-Job monitor, 1-6  
   Memory Maps, 2-8  
 RUBOUT, 2-13, 3-3  
 Rules for user-defined and macro symbols, 5-9  
 RUN command, 2-32  
 Running the program, 8-12  
 Run-time area of memory, 6-10  
 Run-time overlay handler, 6-12  
 Run-time overlay handlers and tables, 6-1  
 Save buffer, 3-10  
 SAVE command  
   (Monitor), 2-30  
   (Editor), 3-10, 3-24  
 Save image, 6-1, 6-2  
   file (SAV), 6-5  
   format, 2-30, 6-21  
 .SAVESTATUS request, 9-77  
 .SBTTL directive, 5-34  
 .SCROL, N-10  
 Scroller, 2-15, 3-29, N-2  
   buffer, N-3  
   logic, N-3  
 .SDAT request, 9-80  
 .SDATC request, 9-81  
 .SDATW request, 9-81  
 Search,  
   algorithm (ODT), 8-24  
   commands (EDIT), 3-4, 3-18, B-6  
   effective address, 8-15  
   limit (ODT), 8-15  
   word, 8-15  
 Searches, 8-14, 8-24  
 Segment boundaries, 4-18  
 Send Data/Receive Data, 1-7  
 Sentinel file, 4-4  
 Separating and delimiting characters, 5-6  
 Separator, 2-11  
 Sequence numbers, 4-7  
 Sequential operations, H-9  
 .SERR request, 9-53  
 Set Bottom Address, L-4  
 SET command, 2-23  
   Options, 2-23  
 Set Relocation Registers, L-4  
 Setting the Editor to immediate mode, 3-30  
 Setting up interrupt vectors, H-3  
 .SETTOP request, 9-82  
 .SET USR NOSWAP command, 9-84  
 Seven-word status buffer, N-10  
 .SFPA request, 9-84  
 Sharing a location, 9-32  
 Sharing system resources, 1-1  
 Single absolute section, 5-54  
 Single instruction,  
   address, 8-14  
   mode, 8-14  
 Single-job,  
   monitor, 2-18, 3-30, 9-7  
   op, 1-1  
 Single load module, 7-1  
 Single operand instructions, C-8  
 Size specification, 9-37  
 Slash, 8-6  
 Soft error recovery, 9-54  
 Software,  
   components, monitor, 2-7  
   mode, H-6  
   name register, N-7



Software identification numbers, xiv  
 Solid state memory, 1-1  
 Source compare (SRCCOM), 1-4, K-1  
   assembling and linking, A-20  
   error messages, K-5  
   switches, K-2, B-16  
 Source-Double Register, C-17  
 Source,  
   code, 1-2  
   code, macro-free 10-1  
   field, 9-3  
   lines, 5-2  
   operand mode, 5-25  
   program, 1-2, 5-2  
   program format, 5-2  
   register, C-15  
 Spacing to the end of the tape, 4-6  
 Special characters, 5-64  
 Special key commands, EDIT, 3-2  
 Special function keys, 2-12, B-3  
   CTRL A, 2-12  
   CTRL B, 2-12  
   CTRL C, 2-12  
   CTRL E, 2-12  
   CTRL F, 2-12  
   CTRL O, 2-13  
   CTRL P, 2-13  
   CTRL S, 2-13  
   CTRL U, 2-13  
   CTRL Z, 2-13  
   RUBOUT, 2-13  
 Special hardware, customization, A-11  
 Special mode TT bit, 9-8  
 Specifier,  
   address, M-2  
   value, M-2, M-3  
 Specifying,  
   a 50-cycle clock rate, A-13  
   directory segments, 4-19  
   extra words per directory entry, 4-18  
   the number of RFl1 platters, A-12  
   .SPFUN request, 9-85  
   .SPND request, 9-87  
 SRCCOM, see Source Compare  
   .SRESET request, 9-85  
 Stack, 9-50  
   address, 6-21  
   pointer, 6-21, 9-7  
   size, changing, 2-1  
 Standard end-of-file card, 9-10  
   .START, N-10  
 Start address, 2-3, 6-7, 9-7  
 START Command, 2-1  
 Start procedure, 2-1  
 Starting and stopping the display processor, M-2  
 Starting block number, 9-18  
   AT, N-10  
 Statement,  
   format, 5-2  
   terminator, 5-2  
 Statements,  
   assembly language, 5-2  
   direct assignment, 5-10, 5-14  
   BASIC/RT-11, F-1  
   .ENDM, 5-74  
   .ENDR, 5-74  
   .MACRO, 5-2  
 Status word, 9-45, H-1  
   .STOP request, N-11  
 Stopping points, 1-3  
 Storage device, 3-1  
 String, BASIC/RT-11, F-5  
 Subconditionals, 5-57  
 Subpicture tag data, N-13  
 Subroutine,  
   call instruction, N-18  
   return, C-14  
 Summary,  
   command, B-1  
   error message, P-1  
   graphics macro calls, N-21  
   MACRO assembler, C-1  
   programmed requests, 9-15  
   switch, B-1  
 SUSPEND command, 2-36  
 Suspending program execution, 9-99  
 Swapped region, 9-9  
 Swapping, 9-56  
   algorithm, 9-9  
 Switch description, 6-18  
 Switches, 2-10  
   CREF, 5-76, 5-78  
   function control, 5-76, 5-78  
   C-23  
   listing control, 5-76, C-23  
   Macro, 5-76,  
   PIP, 4-2  
 Switch summary, B-1  
   CREF, C-24  
   COMI, B-14  
   FILEX, P-15  
   Librarian, B-12  
   Linker, B-11  
   MACRO/CREF, C-23  
   PIF, B-9  
   SRC, B-16  
 Switching between S/J and F/B, A-5  
 Symbol control, 5-54  
 Symbol table, 5-33  
   global, 6-1, 6-5  
   macro, 5-9  
   overflow, 6-23  
   permanent, 5-9  
   switch, 6-23  
   user, 5-9  
   user-defined, 5-3  
 Symbol, value, 5-9  
 Symbolic arguments, 5-37



- Symbols, 5-9
  - entry, 6-5
  - external, 5-10, 6-5
  - global, 5-10, 5-55, 6-5, M-3
  - internal, 5-10
  - local, 5-12, 5-66
  - macro, 5-9
  - multiply-defined, 5-13
  - permanent, 5-9
  - register, 5-11, 5-12
  - user-defined, 5-3, 5-9
- Symbols and expressions, 5-5
- .SYNC and .NOSYN requests, N-11
- .SYNCH, macro call, 9-22
- Synchronization, N-11
- SYSMAC.8K, 9-1, 10-1
- SYSMAC.SML, E-1, 9-1
- System,
  - bootstrap loader, A-6
  - build operation, 2-3, A-1
  - communication, 2-1
  - communication area, 6-6, 9-7
  - concepts, 9-5
  - conventions, 2-3
  - customization, A-1
  - date, 9-11
  - device, 2-1, 2-21, 2-33
  - device scratch blocks, 9-49
  - disk-usage efficiency, 4-12
  - files, 4-10, A-1, A-15
  - handlers, A-1
  - hardware components, 1-5
  - macros, 9-20
  - macro library, A-14, D-1
  - optimization, A-13
  - programs, 1-1
  - software components, 1-3
  - state, 9-21
  - unit, 2-21
- System software components, 1-3
  - ASEMBL, 1-4, 11-1
  - CREF, 1-4, 5-78
  - DUMP, 1-5, I-1
  - EDIT, 1-3, 3-1
  - EXPAND, 1-4, 10-1
  - FILEX, 1-4, J-1
  - Librarian, 1-4, 7-1
  - Linker, 1-4, 6-1
  - MACRO, 1-3, 5-1
  - ODT, 1-5, 8-1
  - PATCH, 1-4, L-1
  - PATCHO, 1-4, M-1
  - PIP, 1-4, 4-1
  - SRCCOM, 1-4, K-1
- TAB character, 3-3, 5-2
- Table of
  - breakpoints, 8-2, 8-13
  - mode forms and modes, 5-25
  - proceed command repeat counts, 5-25
- Tag value, N-7
- Tagged subpicture file structure, N-18, N-20
- Target device, A-5
- Temporary files, 4-23
- Temporary numeric control, 5-49
- Temporary radix control, 5-45
- Tentative entry, 9-47
- Tentative file, 9-13, 9-65
- Terminal,
  - input request, 2-14
  - interrupt, 8-24
- Terminate search, 8-4
- Terminating directives, 5-50
- Terms, 5-17
- Testing patches, 2-27
- Text, 1-2
  - blocks, 7-14
  - buffer, 3-2, 3-21
  - Editor, 3-1
  - mode, 3-2, 3-20
  - modification, 3-4
  - modification commands, 3-20
- TIME command, 2-17
- Time-critical job, 1-6
- Time of day, access to, 9-51
- Time interval, 9-60
- Timed Wait, 1-7
- Timing requests, 9-66
- .TITLE directive, 5-34
- .TLOCK request, 9-91
- TM11 dump mode, A-11
- Trace trap instruction, 8-21
- .TRACK completion routine, N-12
- .TRACK request, N-12
- Tracking object, N-2, N-12, N-19
- Trailing delimiter, 5-43
- Transfer address, 6-7, 6-24
- Transfer,
  - word-for-word, J-2
  - packed image, J-2
- Transferring characters, 9-93
- Transferring files, 4-8, 4-11
  - between RT-11 and DOS/BATCH (or RSTS-11), J-3
  - to RT-11 from DECsystem-10, J-5
- Transferring memory, 2-31
- Transmitting data, 9-68
- TRAP addressing, EMT and, 5-27
- Trap instructions, 8-14, C-12
- Trap interception, 9-92
- .TRAP request, 9-92
- TT,
  - handler for console terminal, H-5
  - printer interrupt, 8-25
- TT.SYS, A-3
- .TTIN request, 9-93
- .TTINR request, 9-94
- .TTVOUT request, 9-95
- .TTOUTR request, 9-95
- Turning off user error interception, 9-95



- .TWAIT request, 9-98
- Two-volume compress, 4-19
- Type ahead, 2-14, H-5
- Types of programmed requests, 9-14
- UIC, User Identification Code, J-3
  - default value, J-4
- Unary operators, 5-8, 5-45, 5-50
- .UNLNK, N-13
- UNLOAD command, 2-21
- .UNLOCK request, 9-57
- Unnamed control section, 6-4
- Unrecoverable hardware/software error, 7-15
- Unresolved globals, 6-1
- Unsave command, 3-25
- Unused areas, 4-16
- Up-arrow construction, 5-63
- Up-arrow, 8-8
  - prompt, H-5
- Update,
  - number, 9-11
  - switch, 7-9
- Updating cassette sequence number, 4-7
- Urgent messages, 9-63
- User command string, 3-1
- User-defined symbol, 5-3, 5-9
  - table, 5-3
- User library searches, 6-16
- User program, 5-3
  - protection, 9-34
- User Service Routine (USR), 2-7
  - address, 2-31
  - swapping, 2-27
- User switch commands and functions (LIBR), 7-2
- User-written device handlers, H-4
- User symbol table, 5-9
- Using,
  - .ASECT Directives, H-3
  - display editor, 3-28
  - libraries, 6-15
  - ODT with F/B jobs, 8-3
  - overlays, 6-10
  - .SETTOP, H-3
  - the system macro library, 9-14
  - the display file handler, N-16
  - the RT-11 system, 1-6
  - the wild-card construction, 4-1
- USR area, 9-11
  - load address, 9-8
- USR
  - ownership acquisition, 9-91
  - swap bit, 9-7
  - swapping, 9-79
- Utility commands, Editor 3-4, 3-24, B-7
- Utility program, PATCH, L-1
- Utility routines, 8-20
- V1 and V2 differences, xv
- V2 additional features, xv
- ..V2.. macro call, 9-24
- Value specifier, M-2, M-3
- Vector addresses, 9-6
- Verify command, 3-15
- Version,
  - number message, 4-21, L-1
  - switch, 4-21
- Vertical formatting, 5-5
- VT-11 Display Handler Library, assembling and building, A-21
- VT-11 Display processor, 1-3, 2-8, 2-15
- VTBASE, N-24
- VTHDLR, N-25
- VTLIB library, N-26
- VTMAC, N-27
- .WAIT request, 9-71, 9-99
- Warning messages, PIP, 4-25
- Wild card,
  - construction, 4-10
  - expansion, 4-10
  - names, J-1
- Word,
  - address, L-3
  - count, H-9
  - for-word transfer, J-2
  - search, 8-15, 8-21
  - status, H-1
- Word count, M-2
- Word count, variable number, 9-68
- .WORD directive, 5-39
- Words, allocating, 2-35
- .WRITC request, 9-101
- Write,
  - end-of-file, H-11
  - file gap, 4-1
  - with extended gap, H-11
- Write command, 3-13
- .WRITE request, 9-100
- .WRITW request, 9-102